**Elasticsearch**

**第一天**

学习目标：

1. 能够理解ElasticSearch的作用（第一章）

2. 能够安装ElasticSearch服务（第二章）

3. 能够理解ElasticSearch的相关概念（第三章）

4. 能够使用java客户端操作ElasticSearch（第四章）

5. 能够理解分词器的作用（第五章）

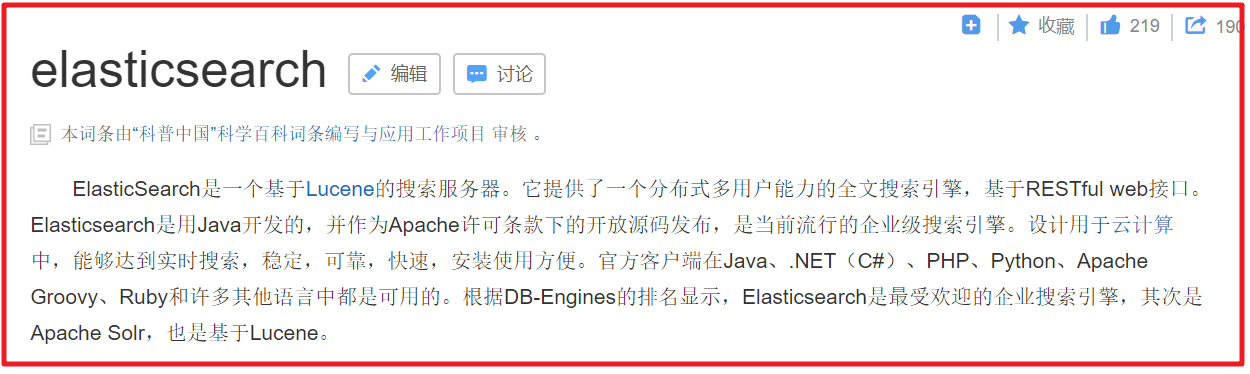
6. 能够使用ElasticSearch集成IK分词器（第五章）

7. 能够使用restful技术操作操作ElasticSearch（附加）

# 第一章：ElasticSearch简介

## 什么是ElasticSearch？

### 介绍



ElasticSearch，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简单。



### 发展历史

2004年Compass诞生。2010年Shay Banon(谢伊●班农)重写Compass，变成著名的Elasticsearch。成为了Github上最活跃的项目之一，拥有超过736名Contributors。2012年Shay 成立商业公司 Elasticsearch 。2014年获得7000万美金融资。2015年，正式更名为Elastic，主要产品从单纯的Elasticsearch变为ELK(Elasticsearch ，Logstash和Kibana)。

2018年，Elastic成功上市，市值达到惊人的50亿美金。目前，Elasticsearch已经有超过3.5亿的产品下载，一个汇聚100,000多名开发人员的社区和超过5500名客户



## ElasticSearch的使用案例

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1300亿行代码”

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、云析分、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

......

## ElasticSearch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，高级功能多由第三方插件提供；

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；

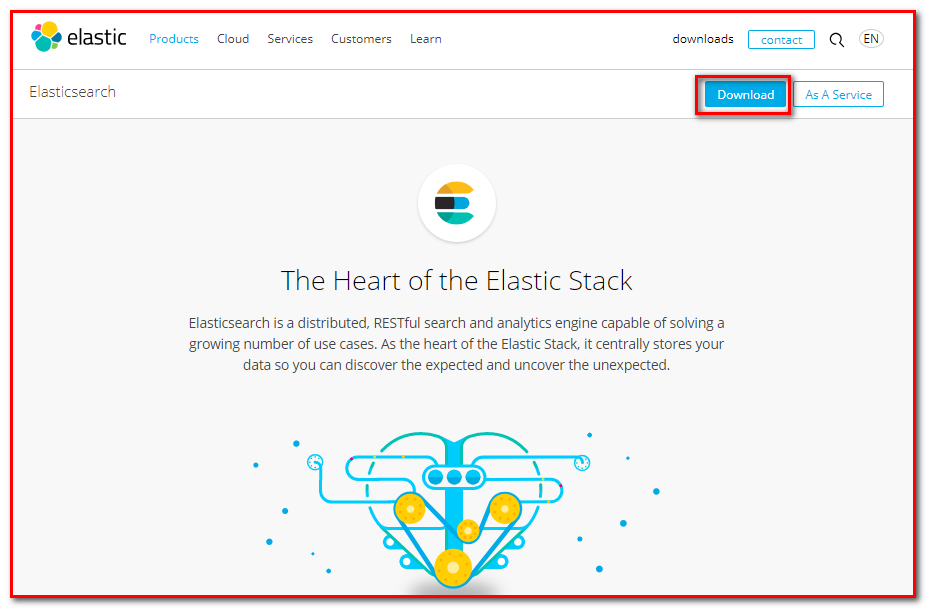
Elasticsearch支持RestFul风格编程（uri的地址，就可以检索），Solr暂不支持；

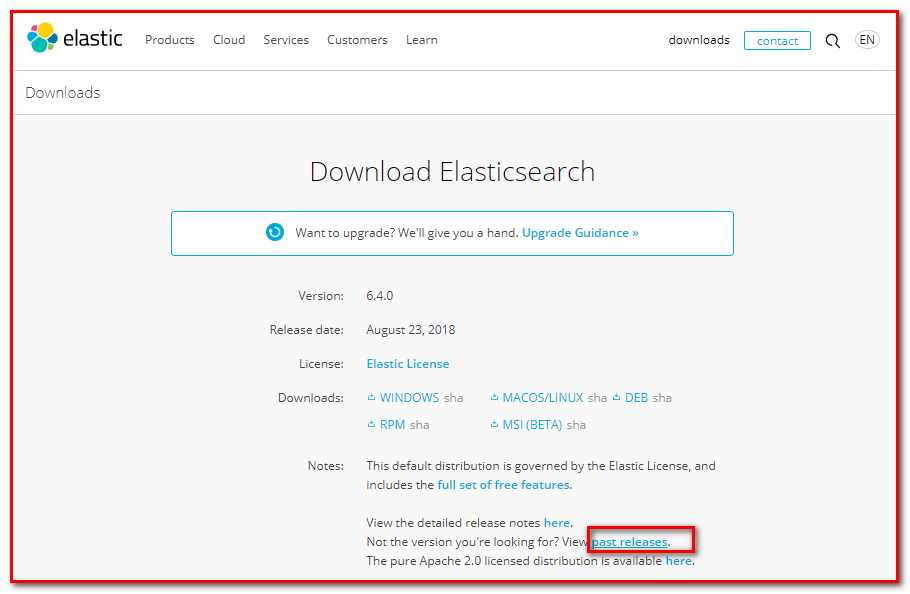
# 第二章：ElasticSearch安装与启动

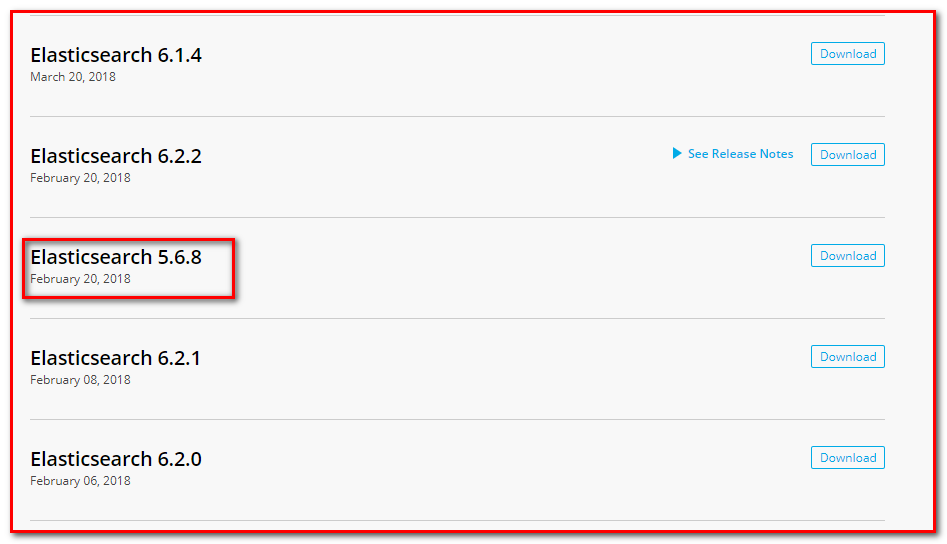
## 下载ES压缩包

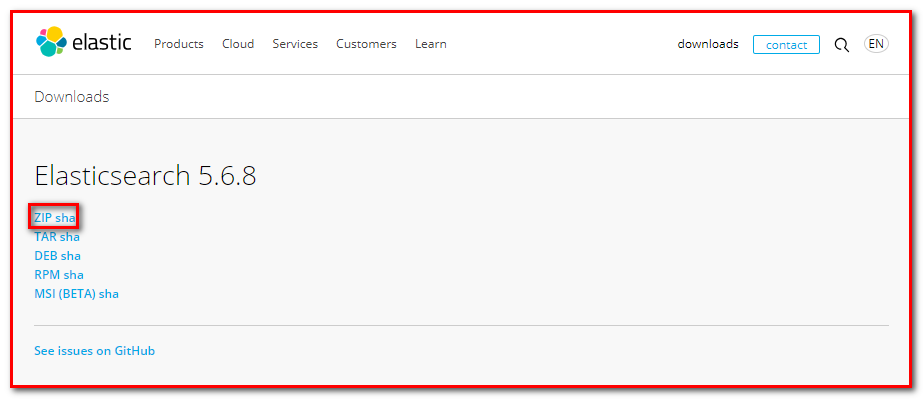
ElasticSearch分为Linux和Window版本，基于我们主要学习的是ElasticSearch的Java客户端的使用，所以我们课程中使用的是安装较为简便的Window版本，项目上线后，公司的运维人员会安装Linux版的ES供我们连接使用。

ElasticSearch的官方地址： <https://www.elastic.co/products/elasticsearch>

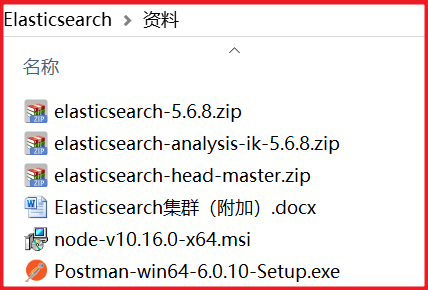








在资料中已经提供了下载好的5.6.8的压缩包：



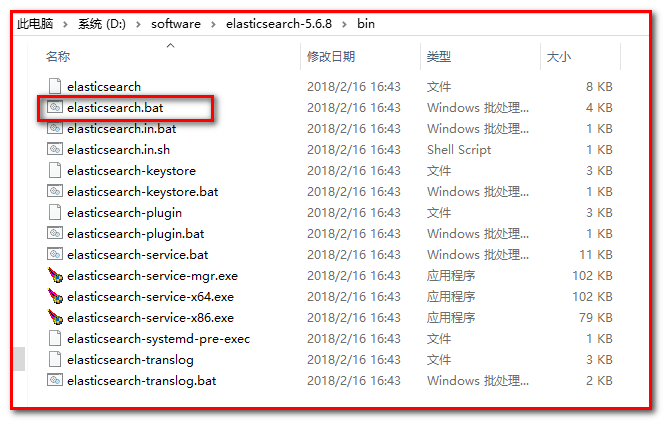
## 安装ES服务

Window版的ElasticSearch的安装很简单，类似Window版的Tomcat，解压开即安装完毕，解压后的ElasticSearch的目录结构如下：

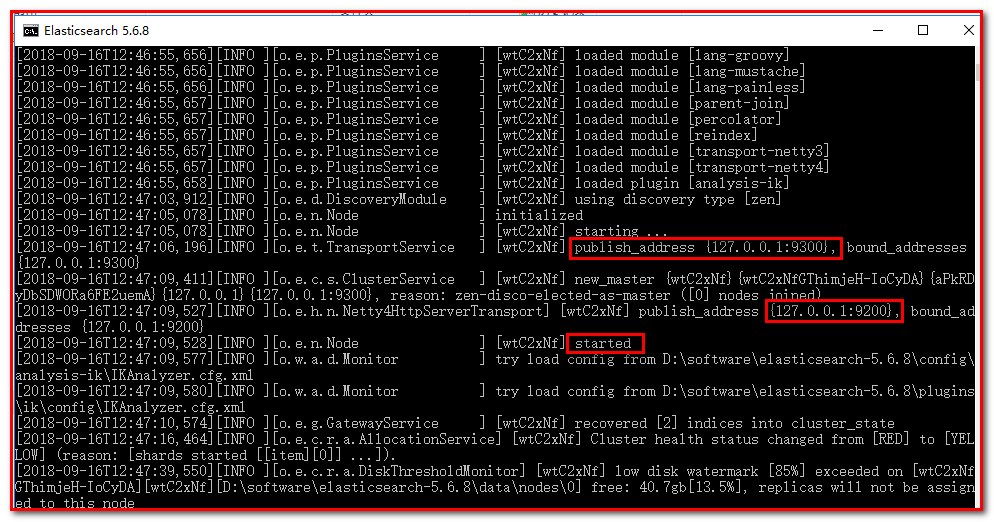
注意：ES的目录不要出现中文，也不要有特殊字符。



## 启动ES服务



启动



注意：

9300是tcp通讯端口，集群间和TCPClient都执行该端口，可供java程序调用；

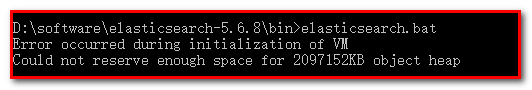
9200是http协议的RESTful接口 。

通过浏览器访问ElasticSearch服务器，看到如下返回的json信息，代表服务启动成功：



注意事项一：ElasticSearch5是使用java开发的，且本版本的es需要的jdk版本要是1.8以上，所以安装ElasticSearch之前保证JDK1.8+安装完毕，并正确的配置好JDK环境变量，否则启动ElasticSearch失败。

注意事项二：出现闪退，通过路径访问发现“空间不足”



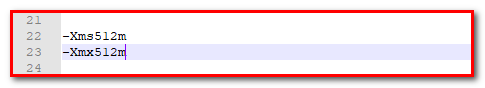
【解决方案】



修改jvm.options文件的22行23行，把2改成1，让Elasticsearch启动的时候占用1个G的内存。



如果1个g还是不行：



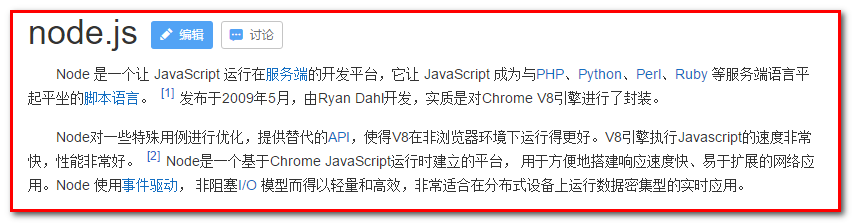
-Xmx521m：设置JVM最大可用内存为512M。  
-Xms512m：设置JVM初始内存为512m。此值可以设置与-Xmx相同，以避免每次垃圾回收完成后JVM重新分配内存。

## 安装ES的图形化界面插件

ElasticSearch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。本课程采用在线的安装。elasticsearch-5-\*以上版本必须安装head需要安装node和grunt。

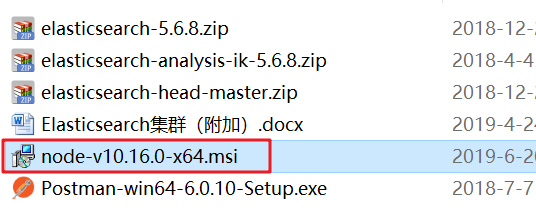
### Nodejs介绍与安装

<http://nodejs.cn/>

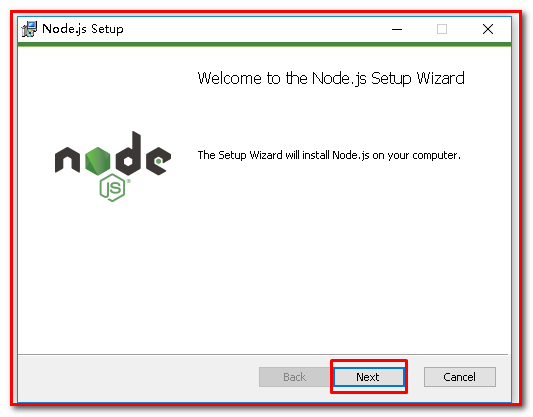


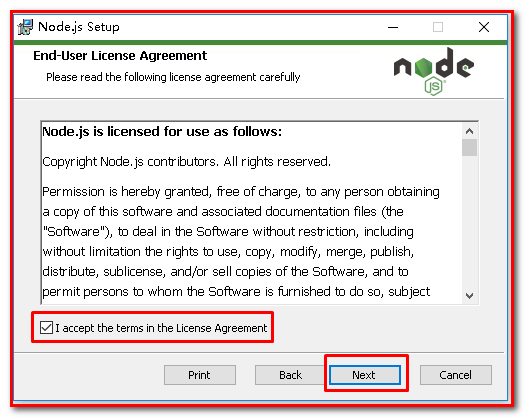
安装es-head需要使用npm命令，要想使用npm命令，需要先安装nodejs。

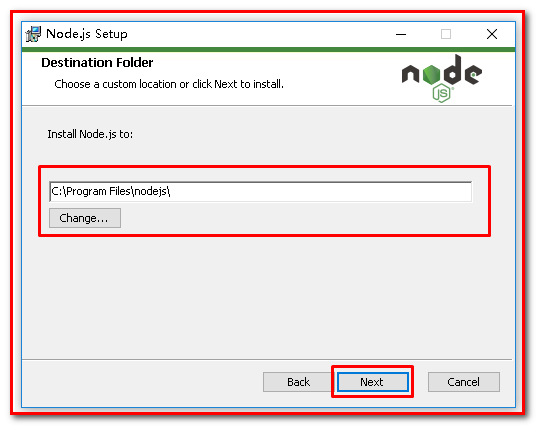
在资料中已经提供了nodejs安装程序：

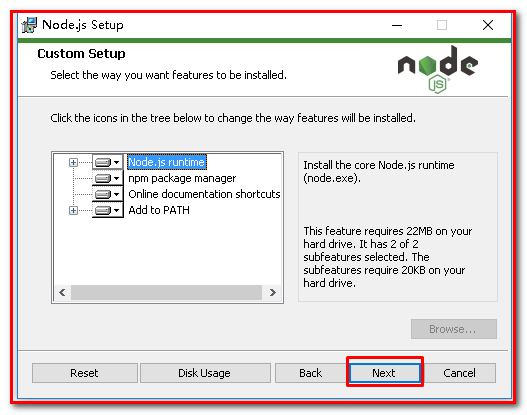


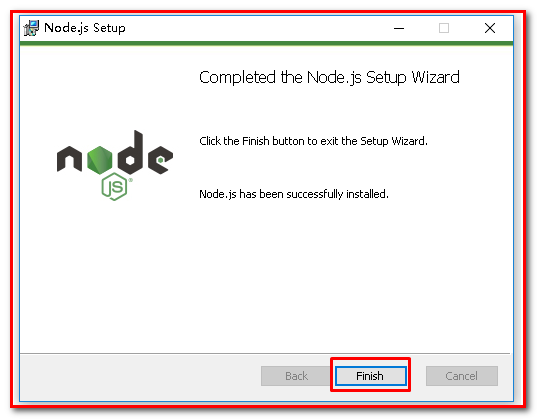
双击安装程序，步骤截图如下：



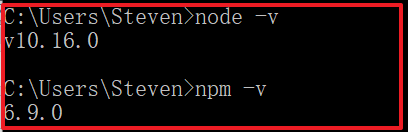








安装完毕，可以通过cmd控制台输入：node -v 查看版本号



### Grunt介绍与安装

<https://www.gruntjs.net/>



#### 正常安装方式

Grunt是基于Node.js的项目构建工具；是基于javaScript上的一个很强大的前端自动化工具，基于NodeJs用于自动化构建、测试、生成文档的项目管理工具。

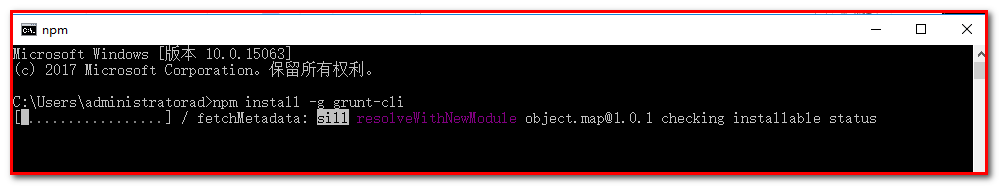
将grunt安装为全局命令，在cmd控制台中输入如下执行命令：

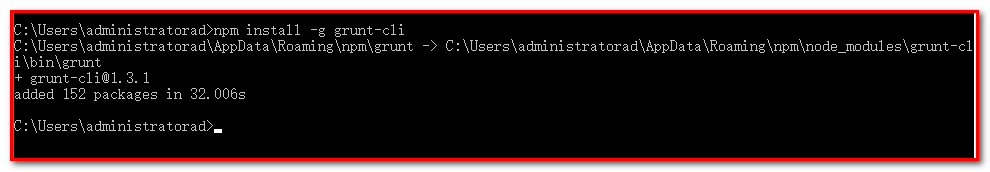
npm install -g grunt-cli

-g表示全局（globle）变量，让grunt-cli的客户端使用全局安装

执行结果如下图：

效果如下：



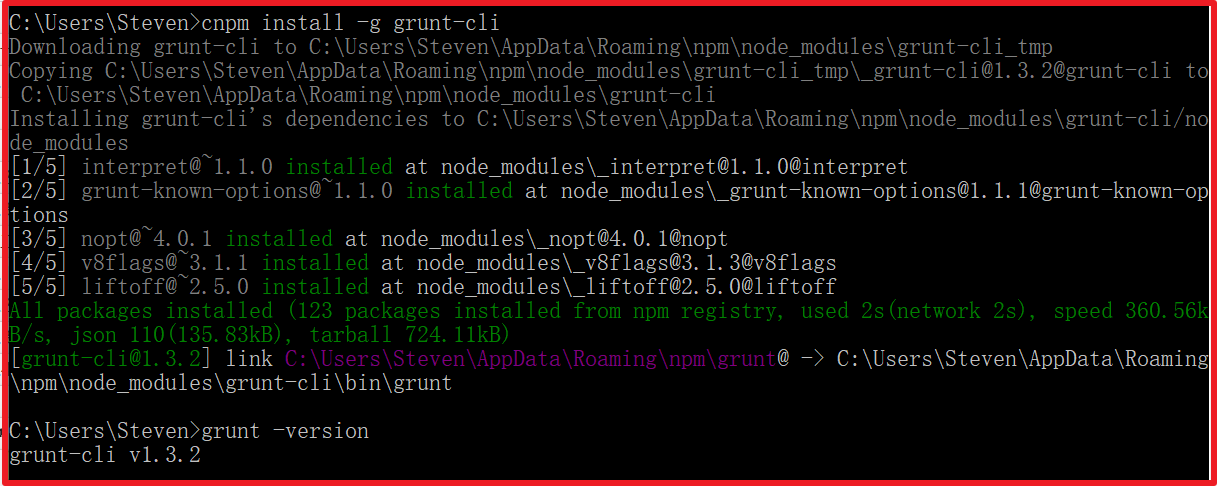


#### 【推荐淘宝镜像安装】

ps:如果安装不成功或者安装速度慢，可以使用淘宝的镜像进行安装：

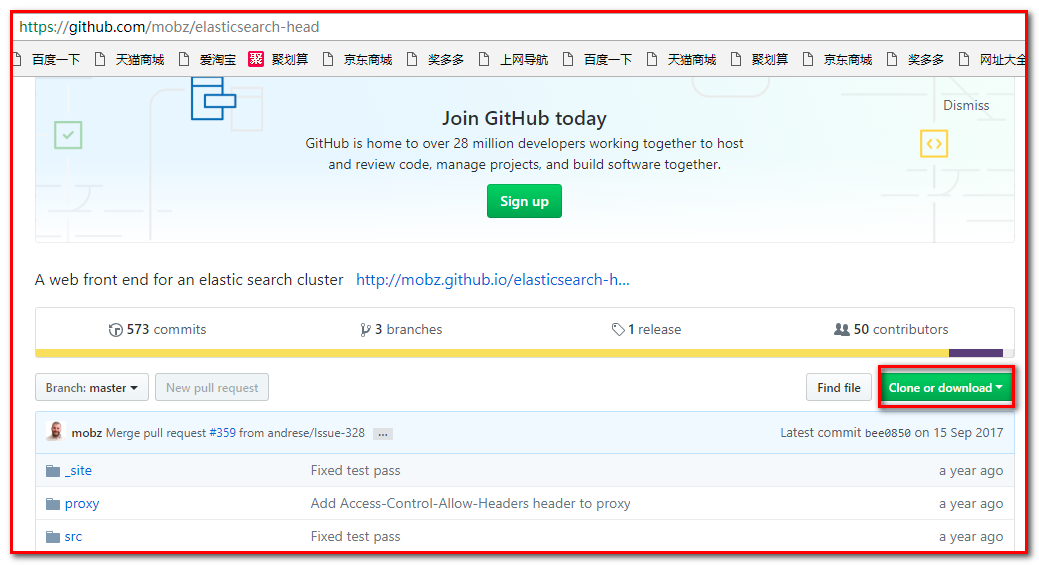
npm install -g cnpm --registry=https://registry.npm.taobao.org

后续使用的时候，只需要把npm xxx 换成 cnpm xxx 即可

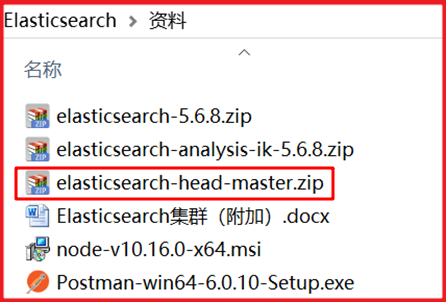


### es-head安装步骤

#### 下载head插件：<https://github.com/mobz/elasticsearch-head>



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



将elasticsearch-head-master压缩包解压到任意目录（我这里是D:\software\elasticsearch-5.6.8），但是要和elasticsearch的安装目录区别开

#### 进入elasticsearch-head-master解压目录执行命令安装

cnpm install

#### 进入head目录启动head，在命令提示符下输入命令：

grunt server



根据提示访问，效果如下：



#### 跨域问题

访问http://localhost:9100/



修改elasticsearch/config下的配置文件：elasticsearch.yml，增加以下三句命令：

http.cors.enabled: true

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

network.host: 127.0.0.1

其中：

http.cors.enabled: true：此步为允许elasticsearch跨域访问，默认是false。

http.cors.allow-origin: "\*"：表示跨域访问允许的域名地址（\*表示任意）。

network.host:127.0.0.1：主机域名

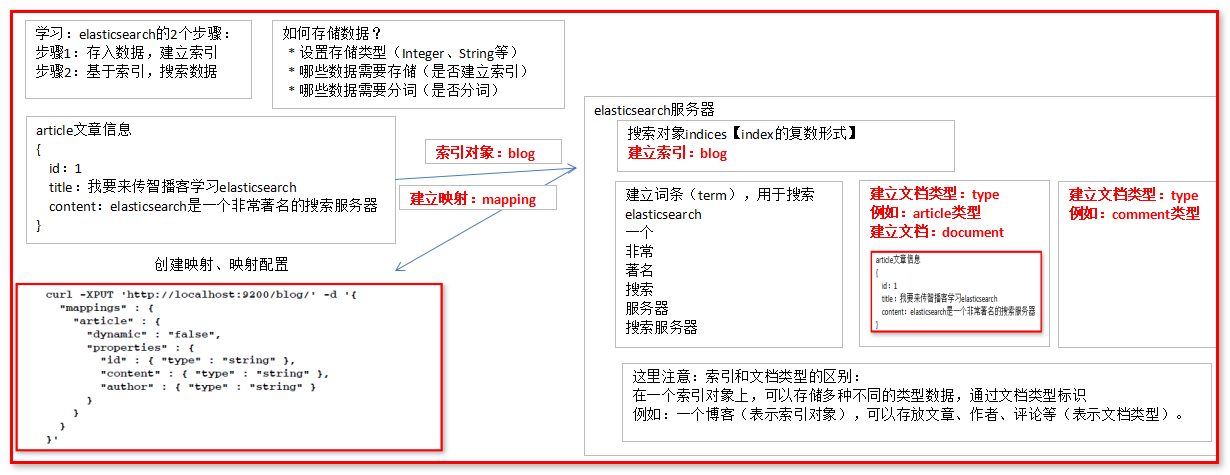
【重启Elasticsearch】，连接成功。

# 第三章：ElasticSearch相关概念(术语)

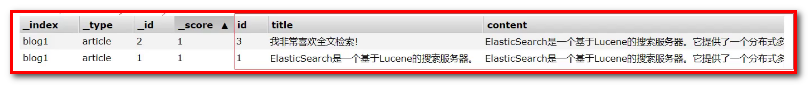
## 概述

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（store），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比较传统关系型数据库如下：

Relational DB -> Databases -> Tables -> Rows -> Columns  
Elasticsearch -> Indices   -> Types -> Documents -> Fields



以下表示2个文档（document），3个字段（field）



## ****Elasticsearch核心概念****

### 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。

### 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。

### 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 Notation）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。

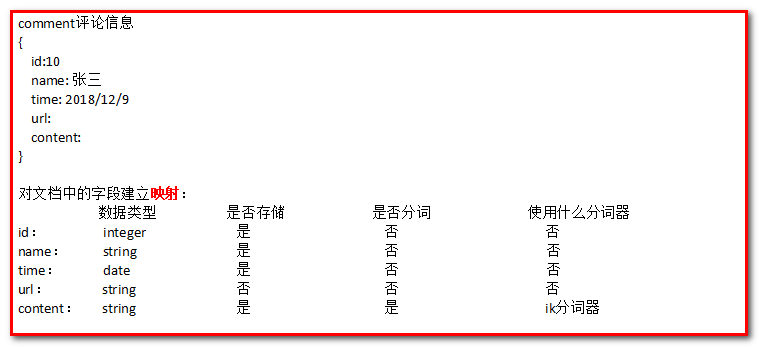
### 域Field

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

### 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、分词器、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。

例如：



### 接近实时 NRT(Near Realtime)

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）

### 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，并一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，因为一个节点只能通过指定某个集群的名字，来加入这个集群

### 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一个指定的集群。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，会默认创建并加入一个叫做“elasticsearch”的集群。

### 分片和复制 shards&replicas

一个索引可以存储超出单个节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。

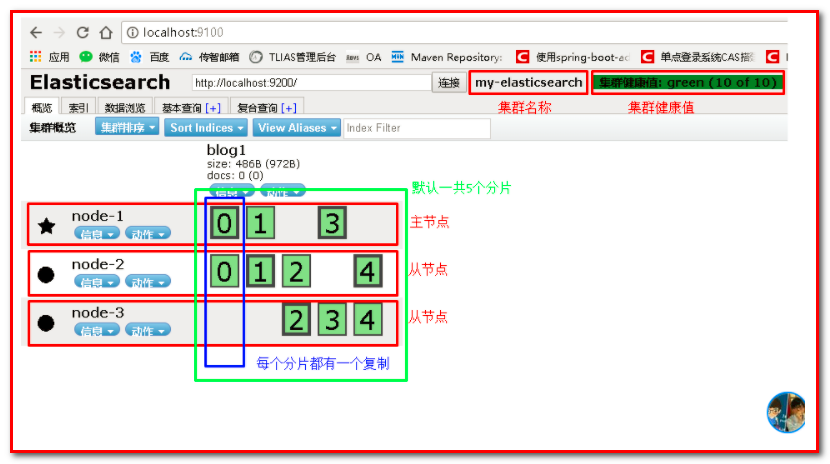
2）允许你在分片（潜在地，位于多个节点上）之上进行分布式的、并行的操作，进而提高性能/吞吐量。

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。

（后面大家自己看即可）

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。一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。

下节课我们会一起搭建集群的环境。

# 第四章：ElasticSearch操作入门

## 创建Maven工程，引入ElasticSearch坐标

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.itheima</**groupId**>  
 <**artifactId**>elasticsearch\_demo</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.elasticsearch</**groupId**>  
 <**artifactId**>elasticsearch</**artifactId**>  
 <**version**>5.6.8</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.elasticsearch.client</**groupId**>  
 <**artifactId**>transport</**artifactId**>  
 <**version**>5.6.8</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.apache.logging.log4j</**groupId**>  
 <**artifactId**>log4j-to-slf4j</**artifactId**>  
 <**version**>2.9.1</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.slf4j</**groupId**>  
 <**artifactId**>slf4j-api</**artifactId**>  
 <**version**>1.7.24</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.slf4j</**groupId**>  
 <**artifactId**>slf4j-simple</**artifactId**>  
 <**version**>1.7.21</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>log4j</**groupId**>  
 <**artifactId**>log4j</**artifactId**>  
 <**version**>1.2.12</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>4.10</**version**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

### 新建索引+添加文档-方式1(map)

使用创建索引（index）+类型（type）+自动创建映射（Elasticsearch帮助我们根据存储的字段自动建立映射，后续讲完分词器后，手动建立映射）

步骤：

/\*\*1.创建客户端访问对象-TransportClient\*/

//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)

//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)

//4.准备索引-client.prepareIndex(索引名,类型名,文档id)

//5.准备数据(域与映射)-indexRequestBuilder.setSource(map)

//6.执行保存索引与文档-indexRequestBuilder.get()

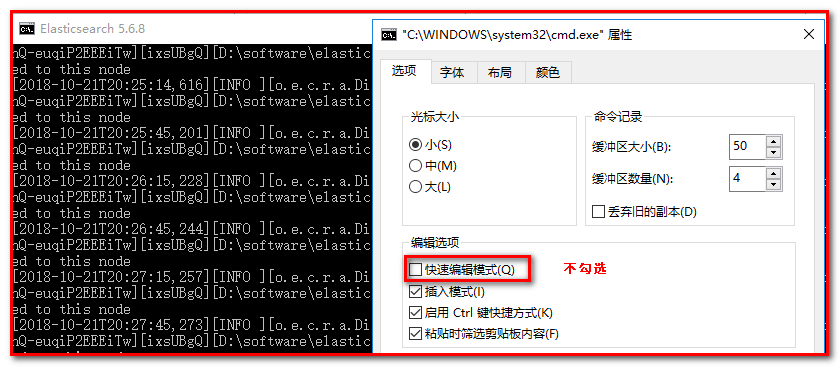
//7.输出执行结果-response.sout()

//8.关闭客户端-client.close()

创建包，com.itheima.es.test，创建类TestElasticSearch.java

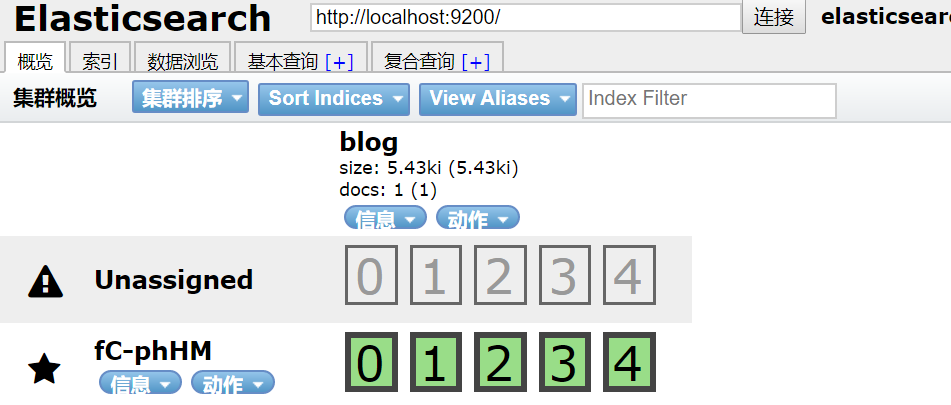
**public class** TestElasticsearch {  
  
 @Test  
 *//创建索引* **public void** testCreateIndex() **throws** Exception{  
 */\*\*  
 \* 1.创建客户端访问对象-TransportClient  
 \* Settings表示集群的设置  
 \* Settings.EMPTY：表示没有集群的配置(单节点)  
 \* Settings.builder().build()：表示连接的是集群环境(多节点)  
 \*/* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
 *//4.准备索引-client.prepareIndex(索引名,类型名,文档id)* IndexRequestBuilder indexRequestBuilder = client.prepareIndex(**"blog"**, **"article"**, **"1"**);  
 *//5.准备数据(域与映射)-indexRequestBuilder.setSource(map)* Map<String, Object> map = **new** HashMap<String, Object>();  
 map.put(**"id"**, 1);  
 map.put(**"title"**, **"什么是Elasticsearch?"**);  
 map.put(**"content"**, **"ElasticSearch是一个基于Lucene的搜索服务器"**);  
 indexRequestBuilder.setSource(map);  
 *//6.执行保存索引与文档-indexRequestBuilder.get()* IndexResponse response = indexRequestBuilder.get();  
 *//7.输出执行结果-response.sout()* System.***out***.println(response);  
 *//8.关闭客户端-client.close()* client.close();  
 }  
}

如果创建索引的时候，插入数据过慢，可以在elasticsearch服务器的上方点击右键->属性，将“快速编辑模式”的复选框的对勾去掉。(一般是设置es内存太小才会慢)



### head插件查看索引信息

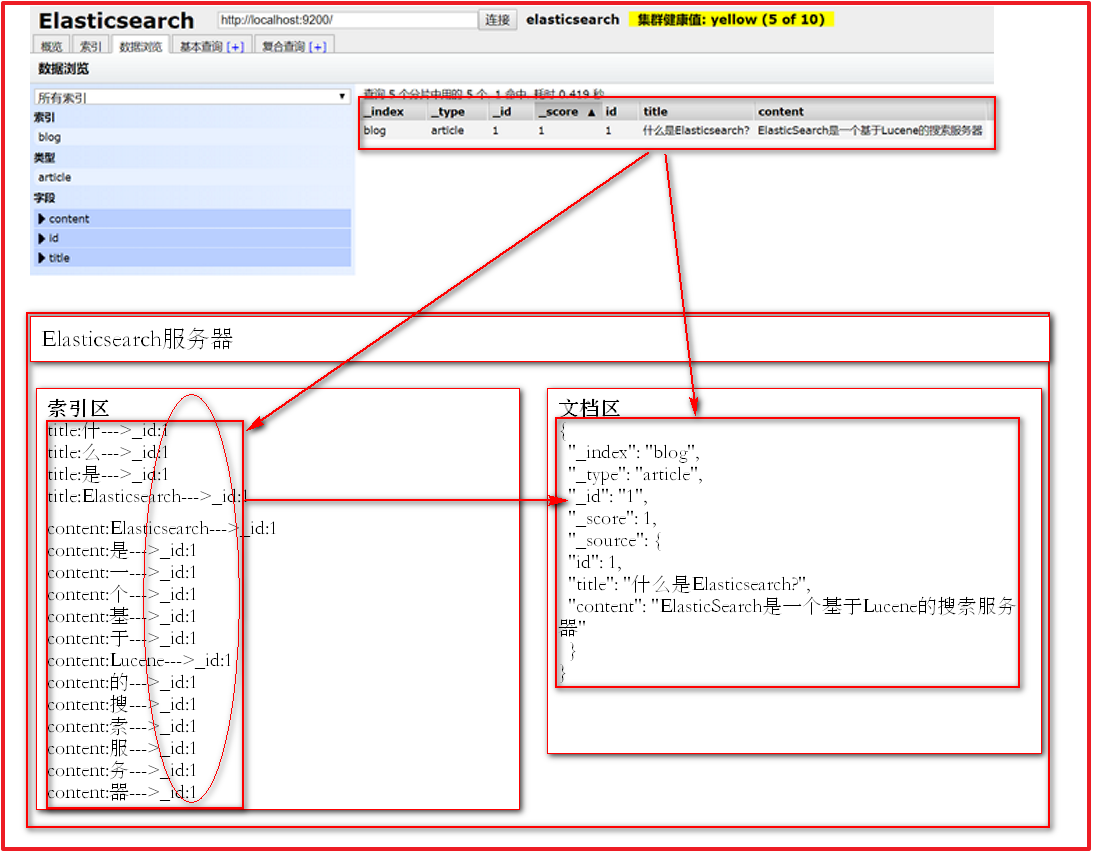
<http://localhost:9100>



查看数据



### Article文章信息在es中存储的内容



### (推荐)新建索引+添加文档-方式2(XContentBuilder)

在原有添加索引代码基础上修改第4步准备索引与第5步准备数据(域与映射)

@Test  
*//创建索引***public void** testCreateIndex() **throws** Exception{  
 */\*\*  
 \* 1.创建客户端访问对象-TransportClient  
 \* Settings表示集群的设置  
 \* Settings.EMPTY：表示没有集群的配置(单节点)  
 \* Settings.builder().build()：表示连接的是集群环境(多节点)  
 \*/* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
  
 *//方式1：map  
 //4.准备索引-client.prepareIndex(索引名,类型名,文档id)  
 /\*IndexRequestBuilder indexRequestBuilder = client.prepareIndex("blog", "article", "1");  
 //5.准备数据(域与映射)-indexRequestBuilder.setSource(map)  
 Map<String, Object> map = new HashMap<String, Object>();  
 map.put("id", 1);  
 map.put("title", "什么是Elasticsearch?");  
 map.put("content", "ElasticSearch是一个基于Lucene的搜索服务器");  
 indexRequestBuilder.setSource(map);\*/  
  
 //方式2：XContentBuilder(推荐)  
 //4.准备索引-client.prepareIndex(索引名,类型名,文档id)* IndexRequestBuilder indexRequestBuilder = client.prepareIndex(**"blog"**, **"article"**, **"2"**);  
 XContentBuilder builder = XContentFactory.*jsonBuilder*()  
 .startObject() *//{* .field(**"id"**,2)  
 .field(**"title"**,**"2什么是Elasticsearch"**)  
 .field(**"content"**,**"2ElasticSearch是一个基于Lucene的搜索服务器"**)  
 .endObject(); *//}* indexRequestBuilder.setSource(builder);  
  
 *//6.执行保存索引与文档-indexRequestBuilder.get()* IndexResponse response = indexRequestBuilder.get();  
 *//7.输出执行结果-response.sout()* System.***out***.println(response);  
 *//8.关闭客户端-client.close()* client.close();  
}

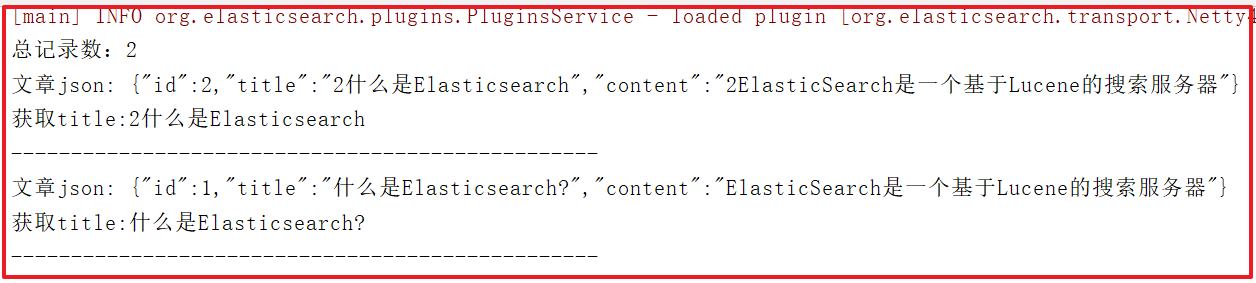
## 搜索文档数据

### ID查询（不走索引）

*//根据文档id查询*@Test  
**public void** testGetById() **throws** Exception{  
 */\*\*  
 \* 1.创建客户端访问对象-TransportClient  
 \* Settings表示集群的设置  
 \* Settings.EMPTY：表示没有集群的配置(单节点)  
 \* Settings.builder().build()：表示连接的是集群环境(多节点)  
 \*/* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
 *//查询* GetResponse response = client.prepareGet(**"blog"**, **"article"**, **"1"**).get();  
 *//7.输出执行结果-response.sout()* System.***out***.println(**"文章json: "** + response.getSourceAsString());  
 System.***out***.println(**"获取title:"** + response.getSource().get(**"title"**));  
 *//8.关闭客户端-client.close()* client.close();  
}

### 查询全部（不走索引）

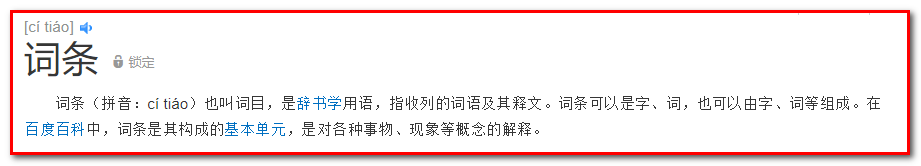
*//查询所有*@Test  
**public void** testGetAll() **throws** Exception{  
 *//1.创建客户端访问对象-TransportClient* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
  
 *//查询所有* SearchResponse response = client.prepareSearch(**"blog"**).setTypes(**"article"**)  
 .setQuery(QueryBuilders.*matchAllQuery*()).get();  
 *//7.输出执行结果-response.sout()* SearchHits hits = response.getHits();  
 System.***out***.println(**"总记录数："** + hits.getTotalHits());  
 **for** (SearchHit hit : hits.getHits()) {  
 System.***out***.println(**"文章json: "** + hit.getSourceAsString());  
 System.***out***.println(**"获取title:"** + hit.getSource().get(**"title"**));  
 System.***out***.println(**"-------------------------------------------------"**);  
 }  
 *//8.关闭客户端-client.close()* client.close();  
}

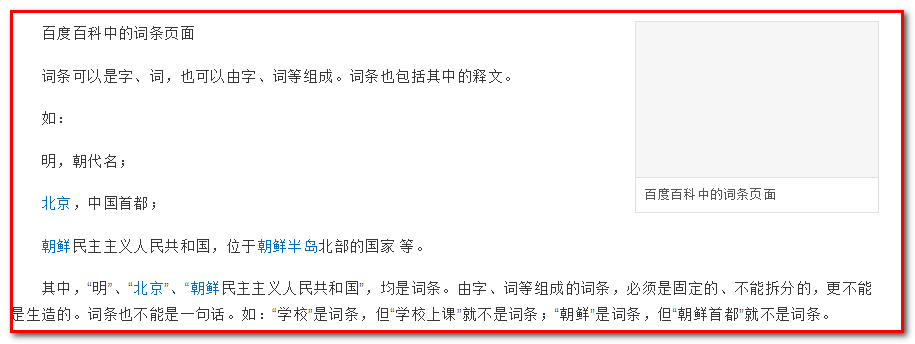


### 字符串查询(条件分词)

*//字符串查询*@Test  
**public void** testGetByString() **throws** Exception{  
 *//1.创建客户端访问对象-TransportClient* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
 *//查询所有* SearchResponse response = client.prepareSearch(**"blog"**).setTypes(**"article"**)  
 */\*QueryBuilders.queryStringQuery("搜索")默认匹配所有域  
 \* 如果如果添加.field("content")：表示只在content字段进行搜索  
 \* \*/* .setQuery(QueryBuilders.*queryStringQuery*(**"搜索"**).field(**"content"**)).get();  
 *//7.输出执行结果-response.sout()* SearchHits hits = response.getHits();  
 System.***out***.println(**"总记录数："** + hits.getTotalHits());  
 **for** (SearchHit hit : hits.getHits()) {  
 System.***out***.println(**"文章json: "** + hit.getSourceAsString());  
 System.***out***.println(**"获取title:"** + hit.getSource().get(**"title"**));  
 System.***out***.println(**"-------------------------------------------------"**);  
 }  
 *//8.关闭客户端-client.close()* client.close();  
}

### 词条查询(条件不分词)





*//字符串查询  
//词条查询*@Test  
**public void** testGetByString() **throws** Exception{  
 *//1.创建客户端访问对象-TransportClient* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
  
 *//查询所有* SearchResponse response = client.prepareSearch(**"blog"**).setTypes(**"article"**)  
 */\*  
 \* QueryBuilders.queryStringQuery("搜索")默认匹配所有域  
 \* 如果如果添加.field("content")：表示只在content字段进行搜索  
 \* \*/  
 //.setQuery(QueryBuilders.queryStringQuery("搜索").field("content")).get();* .setQuery(QueryBuilders.*termQuery*(**"content"**,**"搜索"**)).get();  
 *//7.输出执行结果-response.sout()* SearchHits hits = response.getHits();  
 System.***out***.println(**"总记录数："** + hits.getTotalHits());  
 **for** (SearchHit hit : hits.getHits()) {  
 System.***out***.println(**"文章json: "** + hit.getSourceAsString());  
 System.***out***.println(**"获取title:"** + hit.getSource().get(**"title"**));  
 System.***out***.println(**"-------------------------------------------------"**);  
 }  
 *//8.关闭客户端-client.close()* client.close();  
}

搜不到结果：为什么？

### 模糊查询（通配符查询）

\*：表示所有的任意的多个字符组成

?：表示1个任意的字符

*//模糊查询（通配符查询）*@Test  
**public void** testWildcardQuery() **throws** Exception{  
 *//1.创建客户端访问对象-TransportClient* TransportClient client = **new** PreBuiltTransportClient(Settings.***EMPTY***);  
 *//2.创建Elasticsearch连接对象-new InetSocketTransportAddress(InetAddress.getByName(地址), 端口号)* InetSocketTransportAddress address = **new** InetSocketTransportAddress(InetAddress.*getByName*(**"127.0.0.1"**), 9300);  
 *//3.指定Elasticsearch连接地址与端口-client.addTransportAddress(address)* client.addTransportAddress(address);  
  
 *//查询所有* SearchResponse response = client.prepareSearch(**"blog"**).setTypes(**"article"**)  
 */\*  
 \* "\*"：表示所有的任意的多个字符组成  
 \* "?"：表示1个任意的字符  
 \* \*/* .setQuery(QueryBuilders.*wildcardQuery*(**"content"**,**"\*?\*"**)).get();  
 *//7.输出执行结果-response.sout()* SearchHits hits = response.getHits();  
 System.***out***.println(**"总记录数："** + hits.getTotalHits());  
 **for** (SearchHit hit : hits.getHits()) {  
 System.***out***.println(**"文章json: "** + hit.getSourceAsString());  
 System.***out***.println(**"获取title:"** + hit.getSource().get(**"title"**));  
 System.***out***.println(**"-------------------------------------------------"**);  
 }  
 *//8.关闭客户端-client.close()* client.close();  
}

### 【总结】

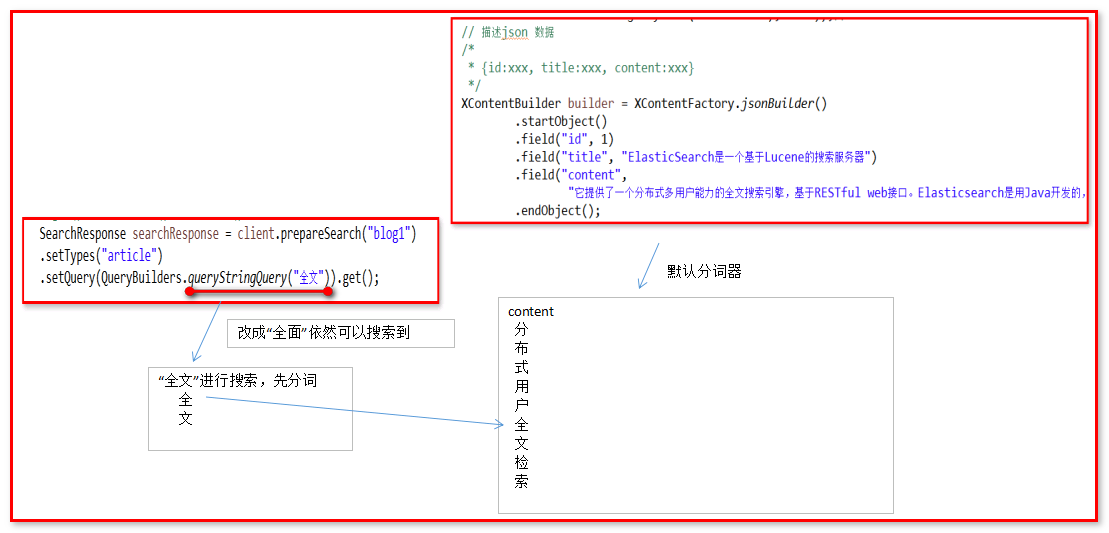
词条： 就是将文本内容存入搜索服务器，搜索服务器进行分词之后的最小的词（不能再切分）

例如：“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、服、务、器

为什么刚才我们使用的QueryStringQuery能搜到数据呢？搜索“全文” 能够查询到，这是为什么呢？看图：



# 第五章：IK 分词器和ElasticSearch集成使用

## 上述查询存在问题分析

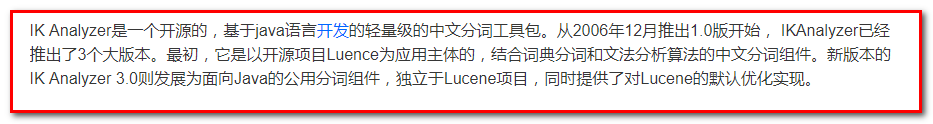
在进行词条查询时，我们搜索"搜索"却没有搜索到数据！

原因：lucene默认是单字分词，在开发中不符合查询的需求，需要定义一个支持中文的分词器。

解决方案：IK分词器

## IK分词器简介

IKAnalyzer是一个开源的，基于java语言开发的轻量级的中文分词工具包。



## ElasticSearch集成IK分词器

### IK分词器的安装

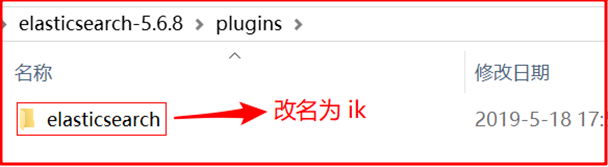
集成IK分词器按照如下步骤来：

1)解压elasticsearch-analysis-ik-5.x后,将文件夹拷贝到elasticsearch-5.6.8\plugins下，并重命名文件夹为ik

2)重新启动ElasticSearch，即可加载IK分词器

下载地址：<https://github.com/medcl/elasticsearch-analysis-ik/releases>





### IK分词器测试

IK提供了两个分词算法ik\_smart 和 ik\_max\_word

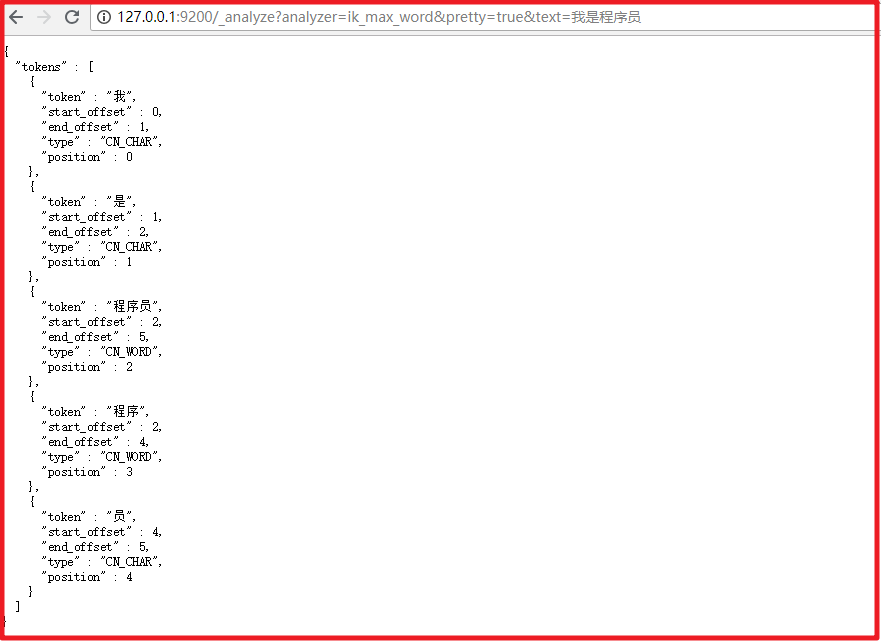
* 最小切分：在浏览器地址栏输入地址

<http://127.0.0.1:9200/_analyze?analyzer=ik_smart&pretty=true&text=我是程序员>



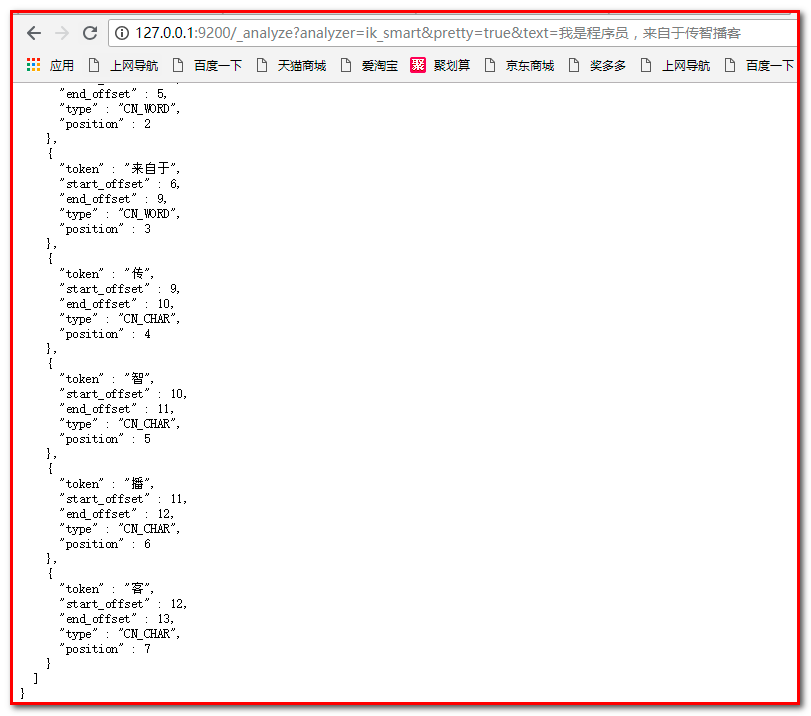
* 最细切分：在浏览器地址栏输入地址

<http://127.0.0.1:9200/_analyze?analyzer=ik_max_word&pretty=true&text=我是程序员>

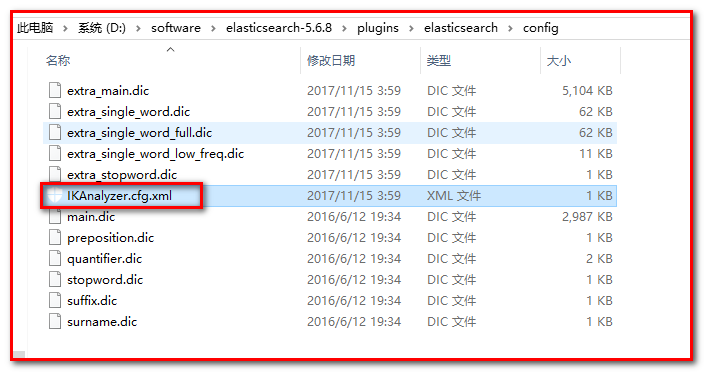


### 对IK分词器，拓展词库

传智播客，默认IK分词是单字分词，想让它做为一个词。



需要配置，IKAnalyzer.cfg.xml：



在当前目录下，添加a.dic作为拓展词库，如果多个拓展词用逗号分开

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">

<properties>

<comment>IK Analyzer 扩展配置</comment>

<!--用户可以在这里配置自己的扩展字典 -->

<entry key="ext\_dict">a.dic</entry>

<!--用户可以在这里配置自己的扩展停止词字典-->

<entry key="ext\_stopwords">b.dic</entry>

<!--用户可以在这里配置远程扩展字典 -->

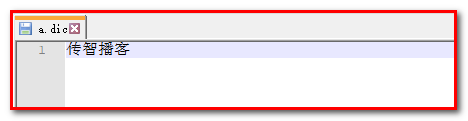
<!-- <entry key="remote\_ext\_dict">words\_location</entry> -->

<!--用户可以在这里配置远程扩展停止词字典-->

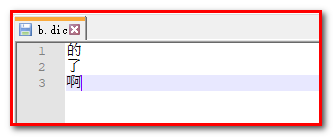
<!-- <entry key="remote\_ext\_stopwords">words\_location</entry> -->

</properties>

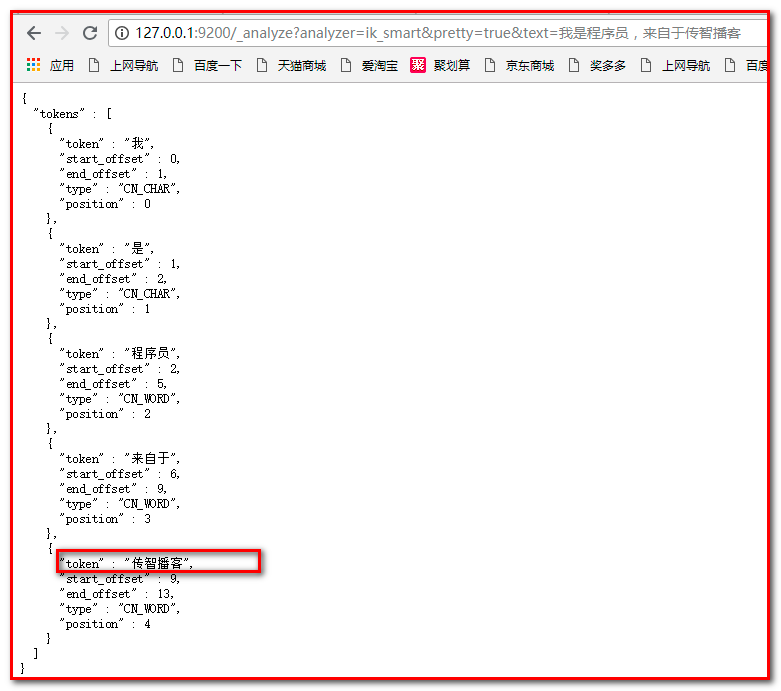
a.dic（使用UTF-8编辑）



b.dic（使用UTF-8编辑）



看到的效果：



### 疑问？

在ElasticSearch没有索引的情况下，插入文档，默认自动创建索引和索引映射是无法使用IK分词器的。

要想让索引库可以使用ik分词器，必须要手动创建映射的配置。如何手动创建映射呢？下回分解。

Java代码手动建立映射（添加ik分词器）（下回分解）

使用restful手动建立映射（添加ik分词器）