

# ElasticSearch第一天

---

## 学习目标：

---

1. 能够理解ElasticSearch的作用
2. 能够安装ElasticSearch服务
3. 能够理解ElasticSearch的相关概念
4. 能够使用java客户端操作ElasticSearch
5. 能够理解分词器的作用
6. 能够使用ElasticSearch集成IK分词器

## 第一章 ElasticSearch简介

---

### 1.1 什么是ElasticSearch

---

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。

### 1.2 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 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本身更侧重于核心功能，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

## 第二章 ElasticSearch安装与启动


---


### 2.1 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>

 [Products](#) [Cloud](#) [Services](#) [Customers](#) [Learn](#)

downloads [contact](#)  [EN](#)


Elasticsearch




## The Heart of the Elastic Stack

Elasticsearch is a distributed, RESTful search and analytics engine capable of solving a growing number of use cases. As the heart of the Elastic Stack, it centrally stores your data so you can discover the expected and uncover the unexpected.



 [Products](#) [Cloud](#) [Services](#) [Customers](#) [Learn](#)

downloads [contact](#)  [EN](#)

Downloads

## Download Elasticsearch

 Want to upgrade? We'll give you a hand. [Upgrade Guidance](#) »

Version:

6.2.3

Release date:

March 20, 2018

Notes:

View the detailed release notes [here](#).  
Not the version you're looking for? View [past releases](#).

Downloads:

 [ZIP](#) sha

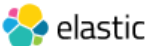
 [TAR](#) sha



 [DEB](#) sha

 [RPM](#) sha

 [MSI \(BETA\)](#) sha

<b>Elasticsearch 6.1.4</b> March 20, 2018	<a href="#">Download</a>
<b>Elasticsearch 6.2.2</b> February 20, 2018	<a href="#">▶ See Release Notes</a> <a href="#">Download</a>
<b>Elasticsearch 5.6.8</b> February 20, 2018	<a href="#">Download</a>


[Products](#)
[Cloud](#)
[Services](#)
[Customers](#)
[Learn](#)

[downloads](#)
[contact](#)



---

Downloads

## Elasticsearch 5.6.8

[ZIP sha](#)
[TAR sha](#)
[DEB sha](#)
[RPM sha](#)
[MSI \(BETA\) sha](#)

[See issues on GitHub](#)

在资料中已经提供了下载好的5.6.8的压缩包：



elasticsearch-5.  
6.8.zip

## 2.2 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：

bin	可执行二进制文件	2018/3/13 10:08	文件夹	
config	配置信息目录	2018/3/13 10:08	文件夹	
lib	jar包存放目录	2018/3/13 10:08	文件夹	
logs	日志存在目录	2018/3/13 10:08	文件夹	
modules	模块存在目录	2018/3/13 10:08	文件夹	
plugins	插件安装目录	2018/3/13 10:08	文件夹	
LICENSE.txt		2018/3/13 10:02	文本文档	12 KB
NOTICE.txt		2018/3/13 10:07	文本文档	188 KB
README.textile		2018/3/13 10:02	TEXTILE 文件	10 KB

## 2.3 启动ES服务

点击ElasticSearch下的bin目录下的elasticsearch.bat启动，控制台显示的日志信息如下：

elasticsearch	2018/3/13 16:15	文件	8 KB
elasticsearch.bat	2018/2/16 16:43	Windows 批处理...	4 KB
elasticsearch.in.bat	2018/3/13 16:17	Windows 批处理...	1 KB
elasticsearch.in.sh	2018/2/16 16:43	SH 文件	1 KB
elasticsearch-keystore	2018/2/16 16:43	文件	3 KB
elasticsearch-keystore.bat	2018/2/16 16:43	Windows 批处理...	1 KB
elasticsearch-plugin	2018/2/16 16:43	文件	3 KB
elasticsearch-plugin.bat	2018/2/16 16:43	Windows 批处理...	1 KB
elasticsearch-service.bat	2018/2/16 16:43	Windows 批处理...	11 KB
elasticsearch-service-mgr.exe	2018/2/16 16:43	应用程序	102 KB
elasticsearch-service-x64.exe	2018/2/16 16:43	应用程序	102 KB
elasticsearch-service-x86.exe	2018/2/16 16:43	应用程序	79 KB
elasticsearch-systemd-pre-exec	2018/2/16 16:43	文件	1 KB
elasticsearch-translog	2018/2/16 16:43	文件	3 KB
elasticsearch-translog.bat	2018/2/16 16:43	Windows 批处理...	2 KB

```

Elasticsearch 5.6.8
[2018-04-04T14:02:40,913][INFO ][o.e.n.Node] JVM arguments [-Xms2g, -Xmx2g, -XX:+UseConcMarkSweepGC, -XX:CMSInitiatingOccupancyFraction=75, -XX:+UseCMSInitiatingOccupancyOnly, -XX:+AlwaysPreTouch, -Xss1m, -Djava.awt.headless=true, -Dfile.encoding=UTF-8, -Djna.nosys=true, -Djdk.io.permissionsUseCanonicalPath=true, -Dio.netty.noUnsafe=true, -Dio.netty.noKeySetOptimization=true, -Dio.netty.recycler.maxCapacityPerThread=0, -Dlog4j.shutdownHookEnabled=false, -Dlog4j2.disable.jmx=true, -Dlog4j.skipJansi=true, -XX:+HeapDumpOnOutOfMemoryError, -Delasticsearch, -Des.path.home=E:\ES\elasticsearch-5.6.8]
[2018-04-04T14:02:41,545][INFO ][o.e.p.PluginsService] [7TD9pkg] loaded module [aggs-matrix-stats]
[2018-04-04T14:02:41,545][INFO ][o.e.p.PluginsService] [7TD9pkg] loaded module [ingest-common]
[2018-04-04T14:02:41,546][INFO ][o.e.p.PluginsService] [7TD9pkg] loaded module [lang-expression]
[2018-04-04T14:02:41,546][INFO ][o.e.p.PluginsService] [7TD9pkg] loaded module [lang-groovy]
[2018-04-04T14:02:41,546][INFO ][o.e.p.PluginsService] [7TD9pkg] loaded module [lang-mustache]
[2018-04-04T14:02:41,546][INFO ][o.e.p.PluginsService] [7TD9pkg] loaded module [lang-painless]
[2018-04-04T14:02:41,546][INFO ][o.e.p.PluginsService] [7TD9pkg] loaded module [parent-join]
[2018-04-04T14:02:41,546][INFO ][o.e.p.PluginsService] [7TD9pkg] loaded module [percolator]
[2018-04-04T14:02:41,546][INFO ][o.e.p.PluginsService] [7TD9pkg] loaded module [reindex]
[2018-04-04T14:02:41,546][INFO ][o.e.p.PluginsService] [7TD9pkg] loaded module [transport-netty3]
[2018-04-04T14:02:41,546][INFO ][o.e.p.PluginsService] [7TD9pkg] loaded module [transport-netty4]
[2018-04-04T14:02:41,547][INFO ][o.e.p.PluginsService] [7TD9pkg] no plugins loaded
[2018-04-04T14:02:42,881][INFO ][o.e.d.DiscoveryModule] [7TD9pkg] using discovery type [zen]
[2018-04-04T14:02:43,261][INFO ][o.e.n.Node] [7TD9pkg] initialized
[2018-04-04T14:02:43,262][INFO ][o.e.n.Node] [7TD9pkg] starting ...
[2018-04-04T14:02:43,852][INFO ][o.e.t.TransportService] [7TD9pkg] publish_address {127.0.0.1:9300}, bound_addresses {127.0.0.1:9300}, [:::1]:9300
[2018-04-04T14:02:46,996][INFO ][o.e.c.s.ClusterService] [7TD9pkg] new_master {7TD9pkg} {7TD9pkg:SfGyC6dB5JG1TA} {7pCQKEszTXq38wDGroJ4fg} {127.0.0.1} {127.0.0.1:9300}, reason: zen-disco-elected-as-master ([0] nodes joined)
[2018-04-04T14:02:47,100][INFO ][o.e.g.GatewayService] [7TD9pkg] recovered [0] indices into cluster state
[2018-04-04T14:02:47,305][INFO ][o.e.h.n.Netty4HttpServerTransport] [7TD9pkg] publish_address {127.0.0.1:9200}, bound_addresses {127.0.0.1:9200}, [:::1]:9200
[2018-04-04T14:02:47,305][INFO ][o.e.n.Node] [7TD9pkg] started

```

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：ElasticSearch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。

## 2.4 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。本文档采用本地安装方式进行head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: <https://github.com/mobz/elasticsearch-head>

在资料中已经提供了head插件压缩包:



elasticsearch-head-master.zip

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

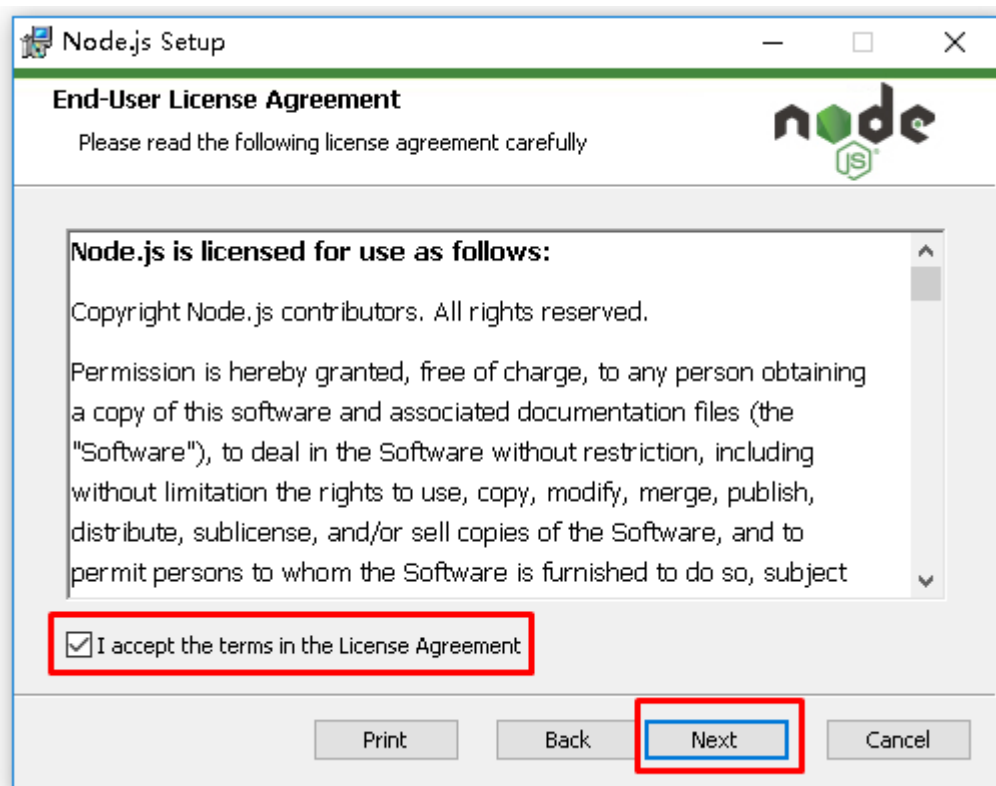
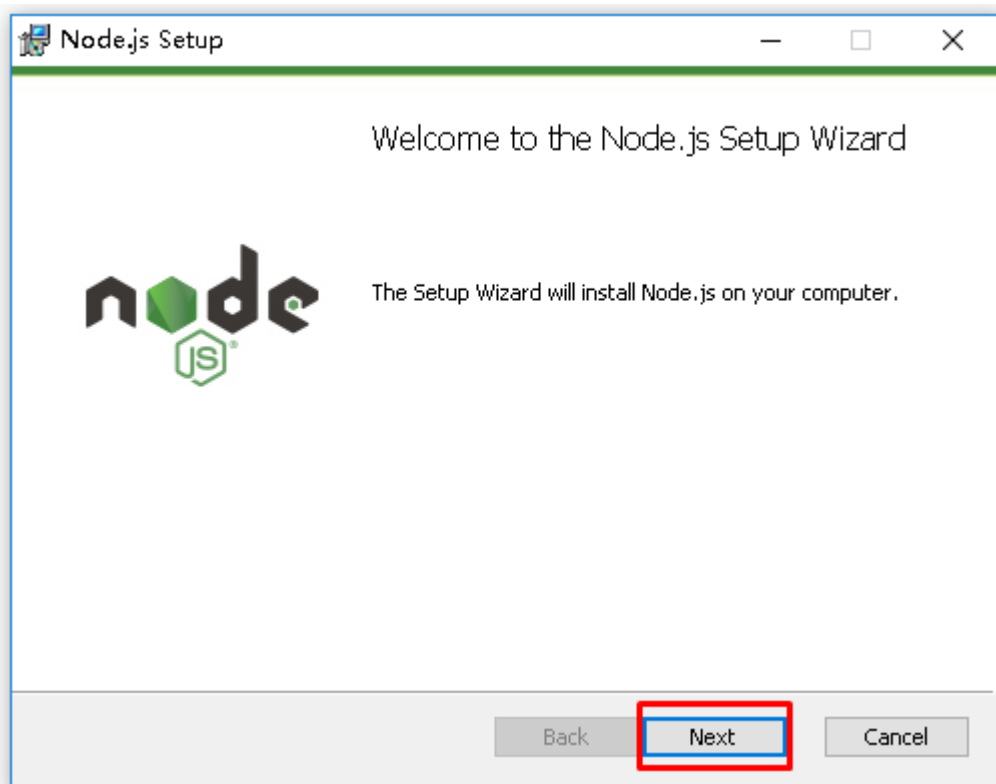
3) 下载nodejs: <https://nodejs.org/en/download/>

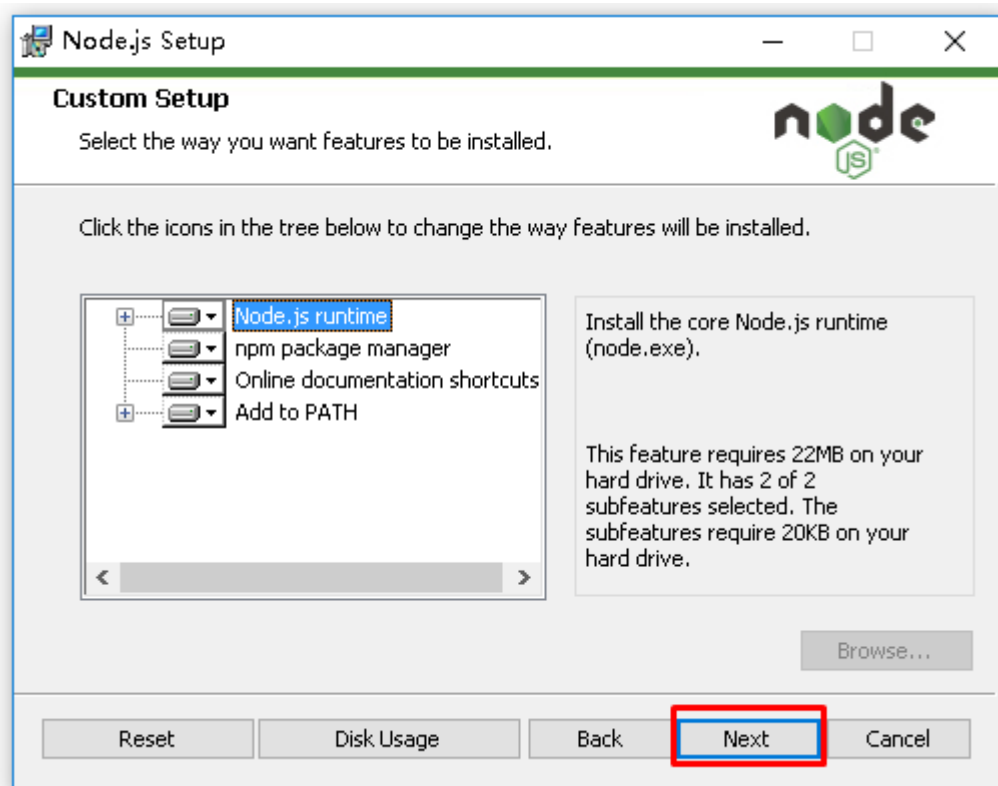
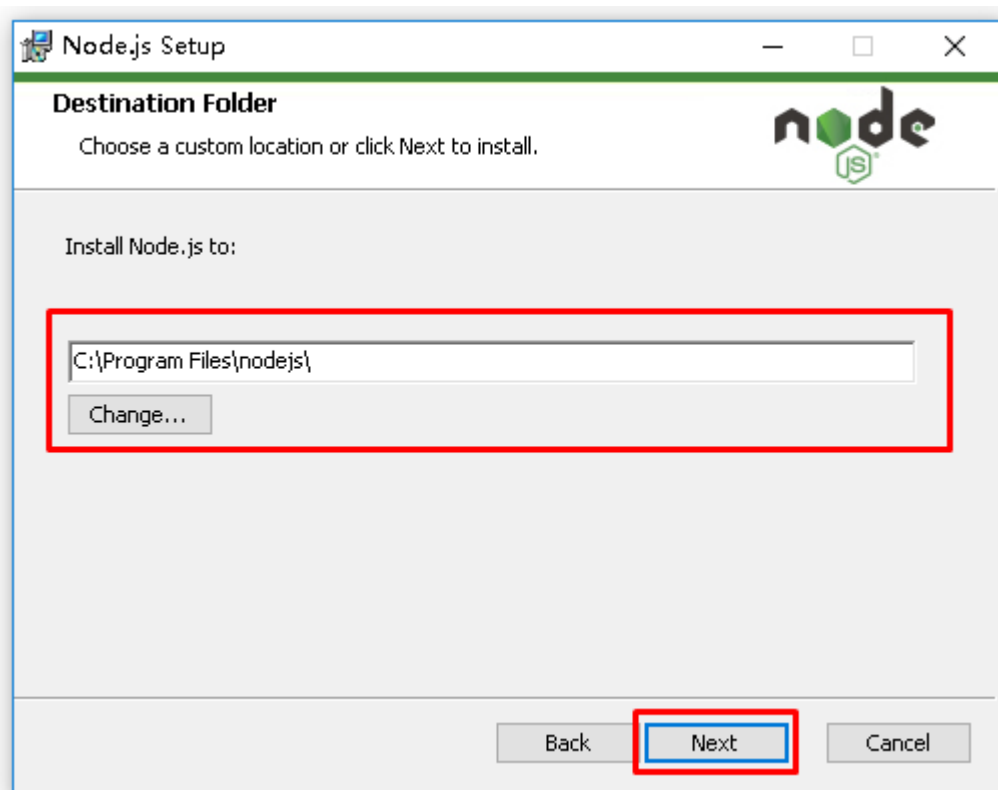
在资料中已经提供了nodejs安装程序:

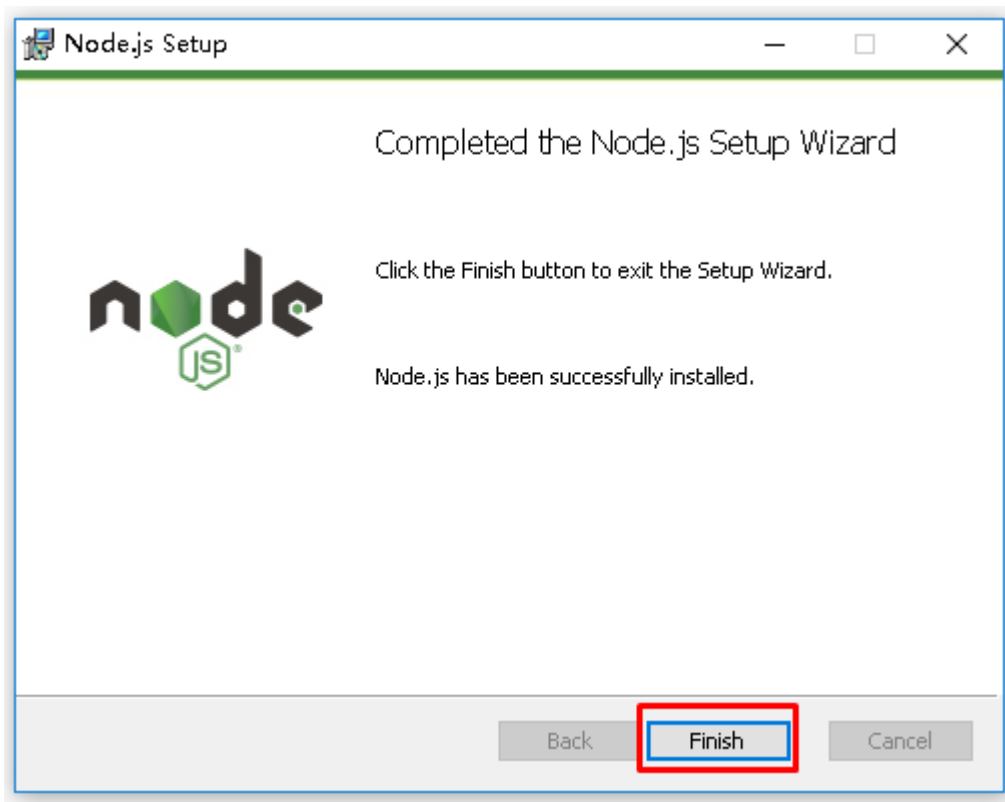


node-v8.9.4-x64.msi

双击安装程序，步骤截图如下:







4) 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
```

执行结果如下图：

```
命令提示符
Microsoft Windows [版本 10.0.16299.309]
(c) 2017 Microsoft Corporation. 保留所有权利。

C:\Users\muzimoo>npm install -g grunt-cli
C:\Users\muzimoo\AppData\Roaming\npm\grunt -> C:\Users\muzimoo\AppData\Roaming\npm\node_modules\grunt-cli\bin\grunt
+ grunt-cli@1.2.0
added 16 packages in 2.578s

C:\Users\muzimoo>
```

5) 修改elasticsearch配置文件：elasticsearch.yml，增加以下两句命令：

```
http.cors.enabled: true
http.cors.allow-origin: ""
```

此步为允许elasticsearch跨越访问

6) 进入head目录启动head，在命令提示符下输入命令：

```
grunt server
```

注：每次使用都要执行



```
ca: grunt
Microsoft Windows [版本 10.0.16299.309]
(c) 2017 Microsoft Corporation。保留所有权利。

C:\Users\muzimoo>cd C:\elasticsearch-head-master

C:\elasticsearch-head-master>grunt server
(node:12764) ExperimentalWarning: The http2 module is an experimental API.
Running "connect:server" (connect) task
Waiting forever...
started connect web server on http://localhost:9100
```

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



## 第三章 Elasticsearch相关概念

### 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。ES类比传统关系型数据库，就像如下：

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

### 3.2 Elasticsearch核心概念

#### 3.2.1 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）

#### 3.2.2 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

#### 3.2.3 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。

### 3.2.4 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。

### 3.2.5 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。

### 3.2.6 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。

### 3.2.7 分片和复制 shards&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。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。

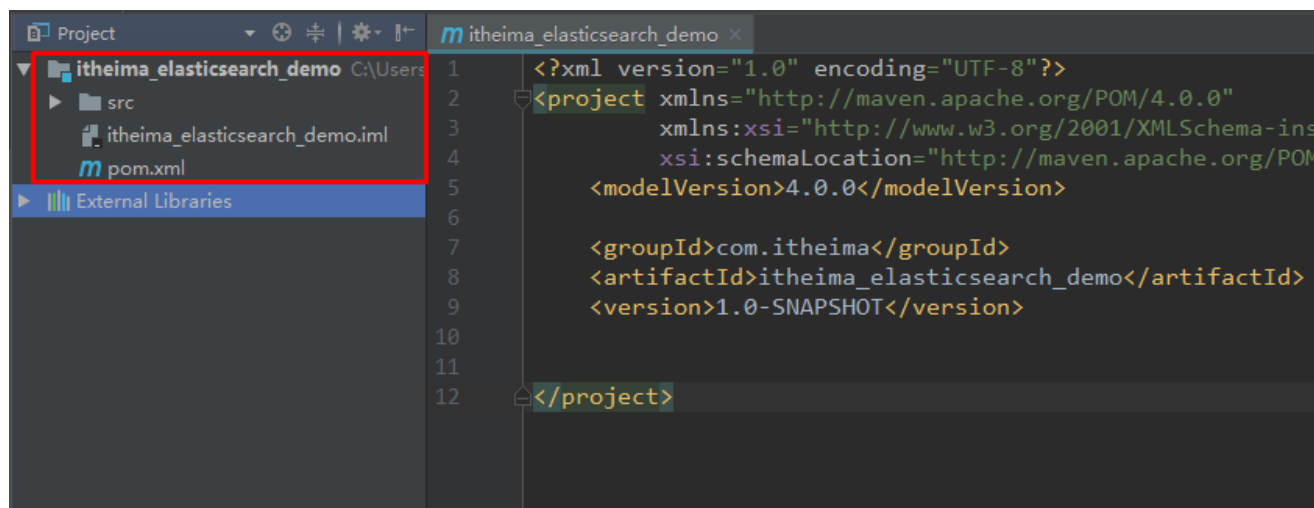
### 3.2.8 映射 mapping

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？和建立表结构表关系数据库三范式类似。

## 第四章 ElasticSearch操作入门

### 4.1 搭建ElasticSearch操作环境

#### 4.1.1 创建Maven工程



#### 4.1.2 导入elasticsearch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>itheima_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>

```

## 4.2 新建索引

### 4.2.1 代码实现

```

@Test
public void test1() throws Exception{
    //1、创建es客户端连接对象
    TransportClient client = new PreBuiltTransportClient(Settings.EMPTY)

        .addTransportAddress(new

```

```

InetSocketAddress(InetAddress.getByName("127.0.0.1"), 9300));

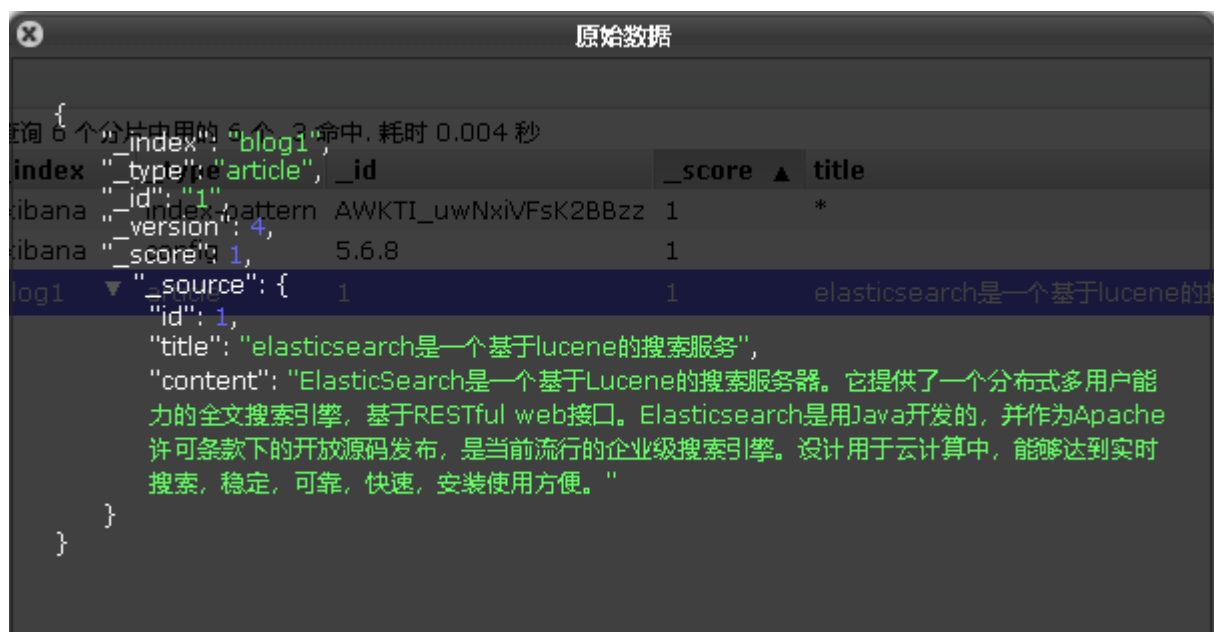
//2、创建文档内容
XContentBuilder builder = XContentFactory.jsonBuilder()
    .startObject()
    .field("id",1)
    .field("title","elasticsearch是一个基于lucene的搜索服务")
    .field("content","ElasticSearch是一个基于Lucene的搜索服务器。" +
        "它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。" +
        "Elasticsearch是用Java开发的，并作为Apache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();
//3、建立文档对象
client.prepareIndex("blog1", "article", "1").setSource(builder).get();
//4、释放资源
client.close();
}

```

## 4.2.2 head插件显示索引信息

The screenshot shows the Elasticsearch head interface. The 'Index Filter' section displays two indices: 'blog1' and '.kibana'. The 'blog1' index is highlighted with a red box. It shows a size of 8.74ki and 1 document. Below the index name, there are two rows of buttons: '0 1 2 3 4' and '0 1 2 3 4'. The 'Unassigned' section shows a star icon and the text '7TD9pkg'.

The screenshot shows the Elasticsearch head interface with the 'Data View' tab selected. A red box highlights the 'Data View' tab and the search result for 'blog1'. The search result table has columns: '\_index', '\_type', '\_id', '\_score', and 'title'. The result for 'blog1' is: '\_index: blog1, \_type: article, \_id: 1, \_score: 1, title: elasticsearch是一个基于lucene的搜索服务'.



## 4.3 搜索文档数据

### 4.3.1 查询全部

#### 4.3.1.1 代码实现

```

@Test
public void test2() throws Exception{
    //1、创建es客户端连接对象
    TransportClient client = new PreBuiltTransportClient(Settings.EMPTY)
        .addTransportAddress(new
    InetSocketAddress(InetAddress.getByName("127.0.0.1"), 9300));

    //2、设置搜索条件
    SearchResponse searchResponse = client.prepareSearch("blog1")
        .setTypes("article").setQuery(QueryBuilders.matchAllQuery())
        .get();

    //3、遍历搜索结果数据
    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"));
    }

    //4、释放资源
    client.close();
}

```

### 4.3.1.2 控制台打印信息

```
查询结果有: 1条
{"id":1,"title":"elasticsearch是一个基于lucene的搜索服务","content":"ElasticSearch是一个基于Lucene的搜索服务器。
title:elasticsearch是一个基于lucene的搜索服务

Process finished with exit code 0
```

## 4.3.2 字符串查询

### 4.3.2.1 代码实现

```
@Test
public void test3() throws Exception{
    //1、创建es客户端连接对象
    TransportClient client = new PreBuiltTransportClient(Settings.EMPTY)
        .addTransportAddress(new
    InetSocketAddress(InetAddress.getByName("127.0.0.1"), 9300));

    //2、设置搜索条件
    SearchResponse searchResponse = client.prepareSearch("blog1")
        .setTypes("article")
        .setQuery(QueryBuilders.queryStringQuery("全垒打")).get();

    //3、遍历搜索结果数据
    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"));
    }

    //4、释放资源
    client.close();
}
```

### 4.3.2.2 控制台打印信息

```
查询结果有: 1条
{"id":1,"title":"elasticsearch是一个基于lucene的搜索服务","content":"ElasticSearch是一个基于Lucene的搜索服务器。
title:elasticsearch是一个基于lucene的搜索服务

Process finished with exit code 0
```

## 4.3.3 词条查询

### 4.3.3.1 代码实现

```
@Test
public void test4() throws Exception{

    //1、创建es客户端连接对象
```

```

TransportClient client = new PreBuiltTransportClient(Settings.EMPTY)
    .addTransportAddress(new
InetSocketAddress(InetAddress.getByName("127.0.0.1"), 9300));

//2、设置搜索条件
SearchResponse searchResponse = client.prepareSearch("blog1")
    .setTypes("article")
    .setQuery(QueryBuilders.termQuery("content", "搜索")).get();

//3、遍历搜索结果数据
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"));
}

//4、释放资源
client.close();
}

```

### 4.3.3.2 控制台打印信息

查询结果有：0条

Process finished with exit code 0

## 4.3.4 模糊查询

### 4.3.4.1 代码实现

```

@Test
public void test5() throws Exception{
    //1、创建es客户端连接对象
    TransportClient client = new PreBuiltTransportClient(Settings.EMPTY)
        .addTransportAddress(new
InetSocketAddress(InetAddress.getByName("127.0.0.1"), 9300));

    //2、设置搜索条件
    SearchResponse searchResponse = client.prepareSearch("blog1")
        .setTypes("article")
        .setQuery(QueryBuilders.wildcardQuery("content", "*全文*")).get();

    //3、遍历搜索结果数据
    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"));
    }

    //4、释放资源
    client.close();

}
```

#### 4.3.4.2 控制台打印信息

查询结果有：0条

Process finished with exit code 0

## 第五章 IK 分词器和ElasticSearch集成使用

### 5.1 上述查询存在问题分析

在进行字符串查询时，我们发现去搜索"全垒打"也可以搜索到数据：

```
SearchResponse searchResponse = client.prepareSearch("blog1")
    .setTypes("article")
    .setQuery(QueryBuilders.queryStringQuery("全垒打")).get();
```

而在进行词条查询时，我们搜索"搜索"却没有搜索到数据：

```
SearchResponse searchResponse = client.prepareSearch("blog1")
    .setTypes("article")
    .setQuery(QueryBuilders.termQuery("content", "搜索")).get();
```

究其原因是ElasticSearch的默认分词器导致的，当我们创建索引时，没有特定的进行映射的创建，所以会使用默认的分词器进行分词，即每个字单独分成一个词。

例如：我是程序员

分词后的效果为：我、是、程、序、员

而我们需要的分词效果是：我、是、程序、程序员

这样的话就需要对中文支持良好的分析器的支持，支持中文分词的分词器有很多，word分词器、庖丁解牛、盘古分词、Ansj分词等，但我们常用的还是下面要介绍的IK分词器。

### 5.2 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。从2006年12月推出1.0版开始，IKAnalyzer已经推出了3个大版本。最初，它是以开源项目Lucene为应用主体的，结合词典分词和文法分析算法的中文分词组件。新版本的IKAnalyzer3.0则发展为 面向Java的公用分词组件，独立于Lucene项目，同时提供了对Lucene的默认优化实现。

IK分词器3.0的特性如下：

1) 采用了特有的“正向迭代最细粒度切分算法”，具有60万字/秒的高速处理能力。2) 采用了多子处理器分析模式，支持：英文字母（IP地址、Email、URL）、数字（日期，常用中文数量词，罗马数字，科学计数法），中文词汇（姓名、地名处理）等分词处理。3) 对中英联合支持不是很好,在这方面的处理比较麻烦.需再做一次查询,同时是支持个人词条的优化的词典存储，更小的内存占用。4) 支持用户词典扩展定义。5) 针对Lucene全文检索优化的查询分析器IKQueryParser；采用歧义分析算法优化查询关键字的搜索排列组合，能极大的提高Lucene检索的命中率。

## 5.3 ElasticSearch集成IK分词器

### 5.3.1 IK分词器的安装

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

课程资料也提供了IK分词器的压缩包：



elasticsearch-analysis-ik-5.6.8.zip

2) 解压，将解压后的elasticsearch文件夹拷贝到elasticsearch-5.6.8\plugins下，并重命名文件夹为ik

 config	2017/11/15 3:59	文件夹	
 commons-codec-1.9.jar	2015/7/2 7:21	Executable Jar File	258 KB
 commons-logging-1.2.jar	2015/7/2 7:21	Executable Jar File	61 KB
 elasticsearch-analysis-ik-5.6.8.jar	2018/3/5 15:25	Executable Jar File	51 KB
 httpclient-4.5.2.jar	2016/8/14 19:32	Executable Jar File	720 KB
 httpcore-4.4.4.jar	2016/8/14 19:32	Executable Jar File	320 KB
 plugin-descriptor.properties	2018/3/5 15:26	PROPERTIES 文件	3 KB

3) 重新启动ElasticSearch，即可加载IK分词器

```
Elasticsearch 5.6.8
[2018-04-06T09:21:37,896][INFO ][o.e.p.PluginsService] [7TD9pkg] loaded module [lang-groovy]
[2018-04-06T09:21:37,896][INFO ][o.e.p.PluginsService] [7TD9pkg] loaded module [lang-mustache]
[2018-04-06T09:21:37,896][INFO ][o.e.p.PluginsService] [7TD9pkg] loaded module [lang-painless]
[2018-04-06T09:21:37,896][INFO ][o.e.p.PluginsService] [7TD9pkg] loaded module [parent-join]
[2018-04-06T09:21:37,896][INFO ][o.e.p.PluginsService] [7TD9pkg] loaded module [percolator]
[2018-04-06T09:21:37,912][INFO ][o.e.p.PluginsService] [7TD9pkg] loaded module [reindex]
[2018-04-06T09:21:37,912][INFO ][o.e.p.PluginsService] [7TD9pkg] loaded module [transport-netty3]
[2018-04-06T09:21:37,912][INFO ][o.e.p.PluginsService] [7TD9pkg] loaded module [transport-netty4]
[2018-04-06T09:21:37,912][INFO ][o.e.p.PluginsService] [7TD9pkg] loaded plugin [analysis-ik]
[2018-04-06T09:21:39,215][INFO ][o.e.d.DiscoveryModule] [7TD9pkg] using discovery type [zen]
[2018-04-06T09:21:39,651][INFO ][o.e.n.Node] [7TD9pkg] initialized
[2018-04-06T09:21:39,651][INFO ][o.e.n.Node] [7TD9pkg] starting ...
[2018-04-06T09:21:40,256][INFO ][o.e.t.TransportService] [7TD9pkg] publish_address {127.0.0.1:9300}, bound_addresses {127.0.0.1:9300}, {[::1]:9300}
[2018-04-06T09:21:43,308][INFO ][o.e.c.s.ClusterService] [7TD9pkg] new_master {7TD9pkg} {7TD9pkgqSfGYc6dB5JG1TA} {I6zuXWIKTdWfIaPZcKkkkIQ} {127.0.0.1} {127.0.0.1:9300}, reason: zen-disco-elected-as-master ([0] nodes joined)
[2018-04-06T09:21:43,371][INFO ][o.w.a.d.Monitor] [7TD9pkg] try load config from C:\elasticsearch-5.6.8\config\analysis-ik\IKAnalyzer.cfg.xml
[2018-04-06T09:21:43,371][INFO ][o.w.a.d.Monitor] [7TD9pkg] try load config from C:\elasticsearch-5.6.8\plugins\ik\config\IKAnalyzer.cfg.xml
[2018-04-06T09:21:43,697][INFO ][o.e.h.n.Netty4HttpServerTransport] [7TD9pkg] publish_address {127.0.0.1:9200}, bound_addresses {127.0.0.1:9200}, {[::1]:9200}
[2018-04-06T09:21:43,697][INFO ][o.e.n.Node] [7TD9pkg] started
[2018-04-06T09:21:43,753][INFO ][o.e.g.GatewayService] [7TD9pkg] recovered [2] indices into cluster_state
[2018-04-06T09:21:43,959][INFO ][o.e.c.r.a.AllocationService] [7TD9pkg] Cluster health status changed from [RED] to [YELLOW] (reason: [shards started [[.kibana][0]] ...)).
```

## 5.3.2 IK分词器测试

IK提供了两个分词算法ik\_smart 和 ik\_max\_word

其中 ik\_smart 为最少切分, ik\_max\_word为最细粒度划分

我们分别来试一下

1) 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](http://127.0.0.1:9200/_analyze?analyzer=ik_smart&pretty=true&text=我是程序员)

输出的结果为:

```
{
  "tokens" : [
    {
      "token" : "我",
      "start_offset" : 0,
      "end_offset" : 1,
      "type" : "CN_CHAR",
      "position" : 0
    },
    {
      "token" : "是",
      "start_offset" : 1,
      "end_offset" : 2,
      "type" : "CN_CHAR",
      "position" : 1
    },
    {
      "token" : "程序员",
      "start_offset" : 2,
```

```
    "end_offset" : 5,  
    "type" : "CN_WORD",  
    "position" : 2  
  }  
]  
}
```

2) 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](http://127.0.0.1:9200/analyze?analyzer=ik_max_word&pretty=true&text=我是程序员)

输出的结果为：

```
{  
  "tokens" : [  
    {  
      "token" : "我",  
      "start_offset" : 0,  
      "end_offset" : 1,  
      "type" : "CN_CHAR",  
      "position" : 0  
    },  
    {  
      "token" : "是",  
      "start_offset" : 1,  
      "end_offset" : 2,  
      "type" : "CN_CHAR",  
      "position" : 1  
    },  
    {  
      "token" : "程序员",  
      "start_offset" : 2,  
      "end_offset" : 5,  
      "type" : "CN_WORD",  
      "position" : 2  
    },  
    {  
      "token" : "程序",  
      "start_offset" : 2,  
      "end_offset" : 4,  
      "type" : "CN_WORD",  
      "position" : 3  
    },  
    {  
      "token" : "员",  
      "start_offset" : 4,  
      "end_offset" : 5,  
      "type" : "CN_CHAR",  
      "position" : 4  
    }  
  ]  
}
```

