**ElasticSearch**（大数据无非解决两个问题：存储+计算！）

版本:ElasticSearch 7.6.1（全网最新了)

6.X,7.X的区别十分大，6.x的API（原生API、RestFul高级!)

**我们要讲解什么?**

SQL : like %狂神说%，如果是的大数据，就十分慢!索引!ElasticSearch:搜索!(百度、 github、淘宝电商!)

1、聊一个人

2、货比三家

3、安装

4、生态圈

5、分词器ik

6、RestFul操作ES

7、CRUD（当成数据库，但比较少）

8、SpringBoot集成ElasticSearch(从原理分析!)

9、爬虫爬取数据！

10、实战，模拟全文搜索

以后你只要，需要用到搜索，就可以使用ES!(大数据量的情况下使用!)后续可以了解一下ELK，做数据分析。

**聊聊Doug Cutting （Hadoop也是这个人创的）**

1998年9月4日，Google公司在美国硅谷成立。正如大家所知，它是一家做搜索引擎起家的公司。



无独有偶，一位名叫Doug Cutting的美国工程师，也迷上了搜索引擎。他做了一个用于文本搜索的函数库（姑且理解为软件的功能组件），命名为Lucene。



Lucene是用JAVA写成的，目标是为各种中小型应用软件加入全文检索功能。因为好用雨且开源(代码公开），非常受程序员们的欢迎。

早期的时候，这个项目被发布在Doug Cutting的个人网站和SourceForge (一个开源软件网站)。后来，2001年底，Lucene成为Apache软件基金会jakarta项目的一个子项目。



2004年，Doug Cutting再接再励，在Lucene的基础上，和Apache开源伙伴Mike Cafarella合作，开发了一款可以代替当时的主搜索的开源搜索引擎，命名为Nutch。

Nutch是一个建立在Lucene核心之上的网页搜索应用程序，可以下载下来直接使用。它在Lucene的基础上加了网络爬虫和一些网页相关的功能，目的就是从一个简单的站内检索推广到全球网络的搜索上，就像Google一样。

Nutch在业界的影响力比Lucene更大。

大批网站采用了Nutch平台，大大降低了技术门槛，使低成本的普通计算机取代高价的Web服务器成为可能。甚至有一段时间，在硅谷有了一股用Nutch低成本创业的潮流。（大数据!)

随着时间的推移，无论是Google还是Nutch，都面临搜索对象"体积"不断增大的问题。

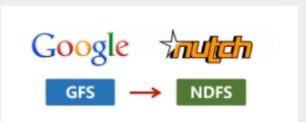
尤其是Google，作为互联网搜索引擎，需要存储大量的网页，并不断优化自己的搜索算法，提升搜索效率。



在这个过程中，Google确实找到了不少好办法，并且无私地分享了出来。

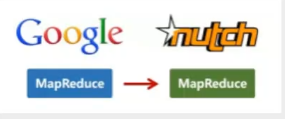
2003年，Google发表了一篇技术学术论文，公开介绍了自己的谷歌文件系统GFS ( Google File System )。这是Google公司为了存储海量搜索数据而设计的专用文件系统。

第二年，也就是2004年，Doug Cutting基于Google的GFS论文，实现了分布式文件存储系统，并将它命名为NDFS ( NutchDistributed File system )。



还是2004年，Google又发表了一篇技术学术论文，介绍自己的MapReduce编程模型。这个编程模型，用于大规模数据集（大于1TB)的并行分析运算。

第二年(2005年)，Doug Cutting又基于MapReduce，在Nutch搜索引擎实现了该功能。



2006年，当时依然很厉害的Yahoo(雅虎）公司，招安了Doug cutting。

  
加盟Yahoo之后，Doug Cutting将NDFS和MapReduce进行了升级改造，并重新命名为Hadoop (NDFS也改名为HDFS ，HadoopDistributed File System )。

这个，就是后来大名鼎鼎的大数据框架系统―—Hadoop的由来。而Doug Cutting，则被人们称为Hadoop之父。

Hadoop这个名字，实际上是Doug Cutting他儿子的黄色玩具大象的名字。所以，Hadoop的Logo，就是一只奔跑的黄色大象。



我们继续往下说。

还是2006年，Google又发论文了。

这次，它们介绍了自己的BigTable。这是一种分布式数据存储系统，一种用来处理海量数据的非关系型数据库。Doug Cutting当然没有放过，在自己的hadoop系统里面，引入了BigTable，并命名为HBase。



好吧，反年就是紧跟Google时代步伐，你出什么，我学什么。所以，Hadoop的核心部分，基本上都有Google的影子。

2008年1月，Hadoop成功上位，正式成为Apache基金会的顶级项目。

同年2月，Yahoo宣布建成了一个拥有1万个内核的Hadoop集群，并将自己的搜索引擎产品部署在上面。7月，Hadoop打破世界纪录，成为最快排序1TB数据的系统，用时209秒。

Lucene是一套信息检索工具包! jar包!不包含搜索引擎系统!包含的︰索引结构!读写索引的工具!排序，搜索规则....工具类!Lucene 和Elasticsearch关系︰

ElasticSearch是基于Lucene做了一些封装和增强（我们上手是十分简单! )

**ElasticSearch概述**

多年前，一个叫做Shay Banon的刚结婚不久的失业开发者，由于妻子要去伦敦学习厨师，他便跟着也去了。在他找工作的过程中，为了给妻子构建一个食谱的搜索引擎，他开始构建一个早期版本的Lucene。

直接基于Lucene工作会比较困难，所以Shay开始抽象Lucene代码以便Java程序员可以在应用中添加搜索功能。他发布了他的第一个开源项目，叫做"Compass"。

后来Shay找到一份工作，这份工作处在高性能和内存数据网格的分布式环境中，因此高性能的、实时的、分布式的搜索引擎也是理所当然需要的。然后他决定重写Compass库使其成为一个独立的服务叫做Elasticsearch,

第一个公开版本出现在2010年2月，在那之后Elasticsearch已经成为Github上最受欢迎的项目之一，代码贡献者超过300人。一家主营Elasticsearch的公司就此成立，他们一边提供商业支持一边开发新功能，不过Elasticsearch将永远开源且对所有人可用。

Shay的妻子依旧等待着她的食谱搜索......

**谁在使用︰**

1、维基百科，类似百度百科，全文检索，高亮，搜索推荐

2、The Guardian(国外新闻网站），类似搜狐新闻，用户行为日志(点击，浏览，收藏，评论）+社交网络数据（对某某新闻的相关看法），数据分析，给到每篇新闻文章的作者，让他知道他的文章的公众反馈(好，坏，热门，垃圾，鄙视，崇拜)

3、Stack Overflow(国外的程序异常讨论论坛），IT问题，程序的报错，提交上去，有人会跟你讨论和回答，全文检索，搜索相关问题和答案，程序报错了，就会将报错信息粘贴到里面去，搜索有没有对应的答案

4、GitHub(开源代码管理），搜索上千亿行代码

5、电商网站，检索商品

6、日志数据分析，logstash采集日志，ES进行复杂的数据分析，ELK技术，elasticsearch+logstash+kibana

7、商品价格监控网站，用户设定某商品的价格阈值，当低于该阈值的时候，发送通知消息给用户，比如说订阅牙膏的监控，如果高露洁牙膏的家庭套装低于50块钱，就通知我，我就去买

8、Bl系统，商业智能，Business Intelligence。比如说有个大型商场集团，Bl，分析一下某某区域最近3年的用户消费金额的趋势以及用户群体的组成构成，产出相关的数张报表，\*\*区，最近3年，每年消费金额呈现100%的增长，而且用户群体85%是高级白领，开一个新商场。ES执行数据分析和挖掘，Kibana进行数据可视化

9、国内︰站内搜索（电商，招聘，门户，等等），IT系统搜索(OA，CRM，ERP，等等），数据分析（ES热门的一个使用场景)

**ES和solr的差别**

**Elasticsearch简介**

Elasticsearch是一个实时分布式搜索和分析引擎。它让你以前所未有的速度处理大数据成为可能。它用于全文搜索、结构化搜索、分析以及将这三者混合使用:I

维基百科使用Elasticsearch提供全文搜索并高亮关键字，以及输入实时搜索(search-asyou-type)和搜索纠错(did-you-mean)等搜索建议功能。

英国卫报使用Elasticsearch结合用户日志和社交网络数据提供给他们的编辑以实时的反馈，以便及时了解公众对新发表的文章的回应。

StackOverflow结合全文搜索与地理位置查询，以及more-like-this功能来找到相关的问题和答案。Github使用Elasticsearch检索1300亿行的代码。

但是Elasticsearch不仅用于大型企业，它还让像DataDog以及Klout这样的创业公司将最初的想法变成可扩展的解决方案。Elasticsearch可以在你的笔记本上运行，也可以在数以百计的服务器上处理PB级别的数据。

Elasticsearch是一个基于Apache Lucene(TM)的开源搜索引擎。无论在开源还是专有领域，Lucene可以被认为是迄今为止最先进、性能最好的、功能最全的搜索引擎库。

但是，Lucene只是一个库。想要使用它，你必须使用ava来作为开发语言并将其直接集成到你的应用中，更糟糕的是，Lucene非常复杂，你需要深入了解检索的相关知识来理解它是如何工作的。

Elasticsearch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。

**Solr简介**

Solr是Apache下的一个顶级开源项目，采用ava开发，它是基于Lucene的全文搜索服务器。Solr提供了比Lucene更为丰富的查询语言，同时实现了可配置、可扩展，并对索引、搜索性能进行了优化

Solr可以独立运行，运行在Jetty、Tomcat等这些Servlet容器中，Solr索引的实现方法很简单，用POST方法向Solr 服务器发送一个描述Field 及其内容的XML文档，Solr根据xml文档添加、删除、更新索引。Solr搜索只需要发送HTTP GET请求，然后对Solr返回Xml.json等格式的查询结果进行解析，组织页面布局。Solr不提供构建UI的功能，Solr提供了一个管理界面，通过管理界面可以查询Solr的配置和运行情况。

solr是基于lucene开发企业级搜索服务器，实际上就是封装了lucene。

Solr是一个独立的企业级搜索应用服务器，它对外提供类似于Web-service的API接口。用户可以通过http请求，向搜索引擎服务器提交一定格式的文件，生成索引;也可以通过提出查找请求，并得到返回结果。

**Lucene简介**

Lucene是apache软件基金会4 jakarta项目组的一个子项目，是一个开放源代码的全文检索引擎工具包，但它不是一个完整的全文检索引擎，而是一个全文检索引擎的架构，提供了完整的查询引擎和索引引擎，部分文本分析引擎（英文与德文两种西方语言)。Lucene的目的是为软件开发人员提供一个简单易用的工具包，以方便的在目标系统中实现全文检索的功能，或者是以此为基础建立起完整的全文检索引擎。Lucene是一套用于全文检索和搜寻的开源程式库，由Apache软件基金会支持和提供。Lucene提供了一个简单却强大的应用程式接口，能够做全文索引和搜寻。在Java开发环境里Lucene是一个成熟的免费开源工具。就其本身而言，Lucene是当前以及最近几年最受欢迎的免费Java信息检索程序库。人们经常提到信息检索程序库，虽然与搜索引擎有关，但不应该将信息检索程序库与搜索引擎相混淆。

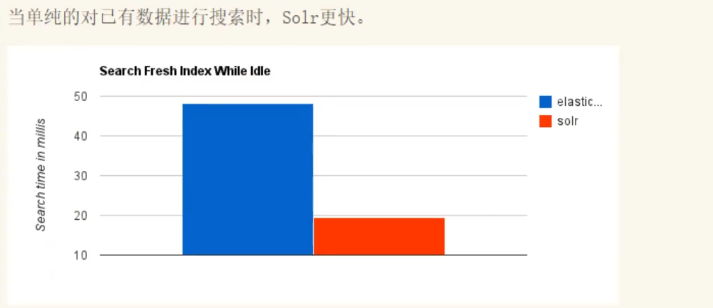
Lucene是一个全文检索引擎的架构。那什么是全文搜索引擎﹖

全文搜索引擎是名副其实的搜索引擎，国外具代表性的有Google、FastU/AllTheWeb、AltaVista、Inktomi、Teoma、WiseNut等，国内著名的有百度（ Baidu ) 。它们都是通过从互联网上提取的各个网站的信息(以网页文字为主)而建立的数据库中，检索与用户查询条件匹配的相关记录，然后按一定的排列顺序将结果返回给用户，因此他们是真正的搜索引擎。

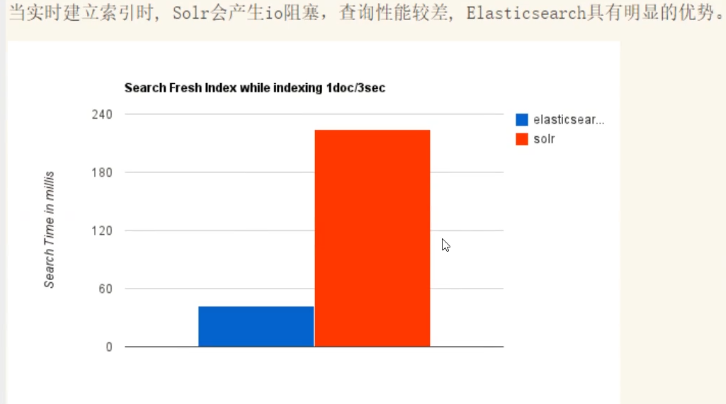
从搜索结果来源的角度，全文搜索引擎又可细分为两种，一种是拥有自己的检索程序( Indexer )，俗称"蜘蛛" ( Spider)程序或\*机器人" ( Robot )程序，并自建网页数据库，搜索结果直接从自身的数据库中调用，如上面提到的7家引擎;另一种则是租用其他引擎的数据库，并按自定的格式排列搜索结果，如Lycos引擎。

**Elasticsearch和solr比较**

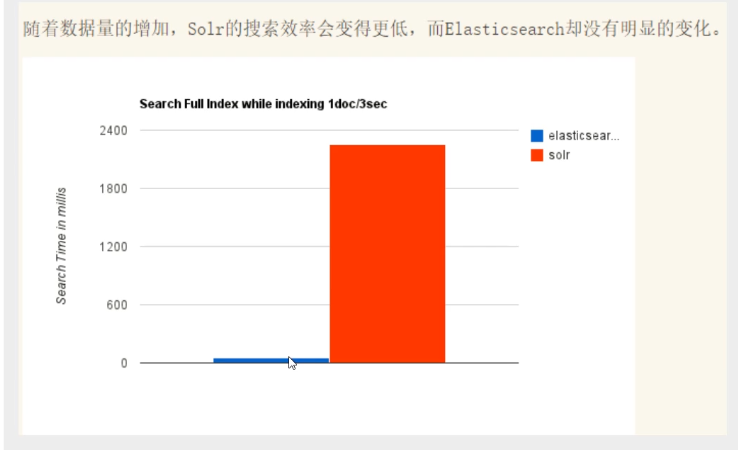
当单纯的对已有数据进行搜索时，Solr更快:

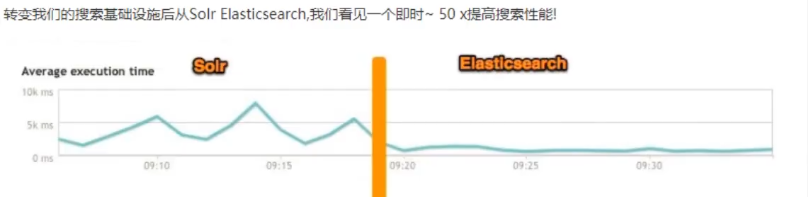


当实时建立索引时，Solr会产生io阻塞，查询性能较差， Elasticsearch具有明显的优势:



随着数据量的增加，Solr的搜索效率会变得更低，而Elasticsearch却没有明显的变化:



转变我们的搜索基础设施后从Solr Elasticsearch,我们看见一个即时~50 x提高搜索性能!

Elastfcsearch vs solr总结

1、es基本是开箱即用（解压就可以用! )，非常简单。Solr安装略微复杂一丢丢!

2、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。3、Solr支持更多格式的数据，比如JSON、XML、CSV，而Elasticsearch 仅支持json文件格式。

4、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，例如图形化界面需要kibana友好支撑~!

5、Solr查询快，但更新索引时慢（即插入删除慢），用于电商等查询多的应用;·ES建立索引快（即查询慢），即实时性查询快，用于facebook新浪等搜索。

. Solr是传统搜索应用的有力解决方案，但Elasticsearch更适用于新兴的实时搜索应用。

1. Solr比较成熟，有一个更大，更成熟的用户、开发和贡献者社区，而Elasticsearch相对开发维护者较少，更新太快，学习使用成本较高。（趋势!)

**Windors下安装：**

**官网下载ES和Kibana：<https://www.elastic.co/>**

1. 解压文件即可



1. 熟悉目录

bin启动文件

config配置文件

log4j2 日志配置文件

jvm.options java虚拟机相关的配置

elasticsearch .yml elasticsearch 的配置文件!默认9200端口!跨域!

lib相关jar包logs日志

modules功能模块

plugins插件!ik分词器

1. 打开bin目录下的elasticsearch.bat，启动访问端口9200

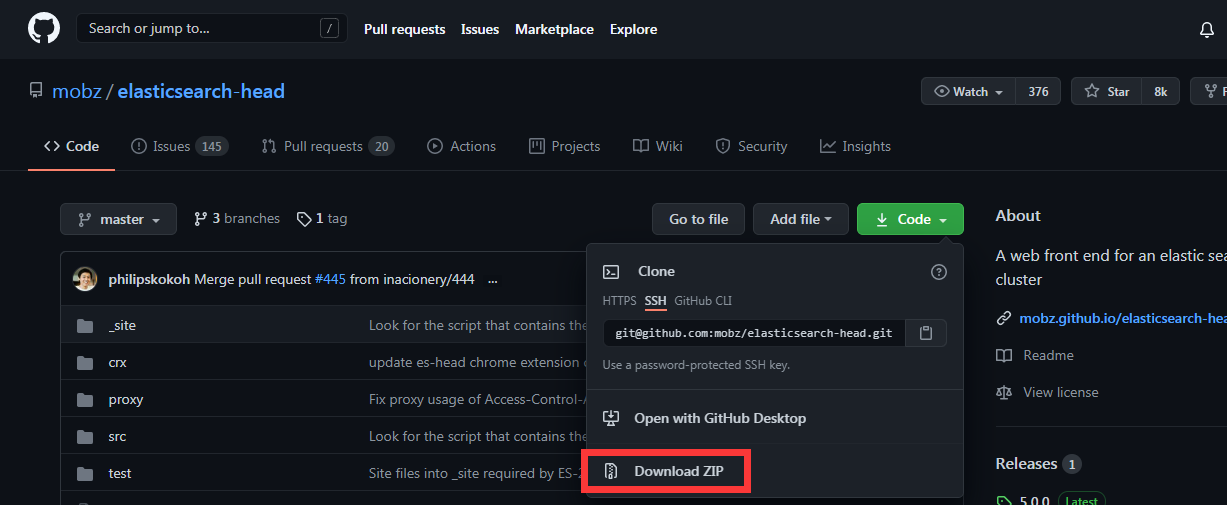


1. 访问测试

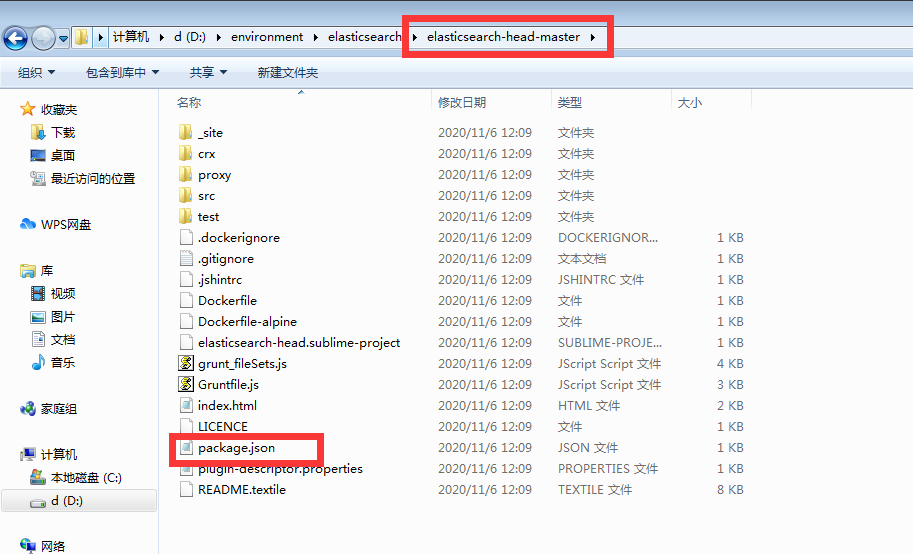


安装可视化界面：elastic-head

1、直接github下载zip解压，<https://github.com/mobz/elasticsearch-head>



1. 到插件文件夹下打开package.json可以看到以来的包，然后再这个页面使用cnpm install

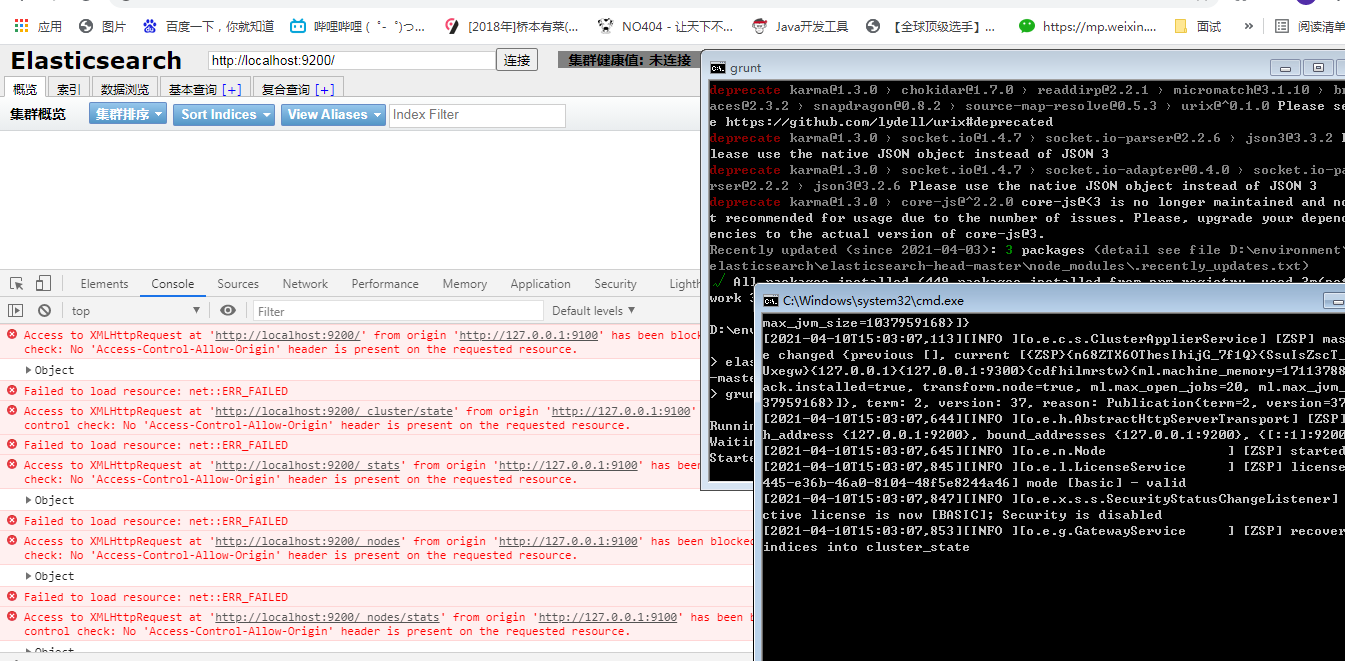


依次输入：

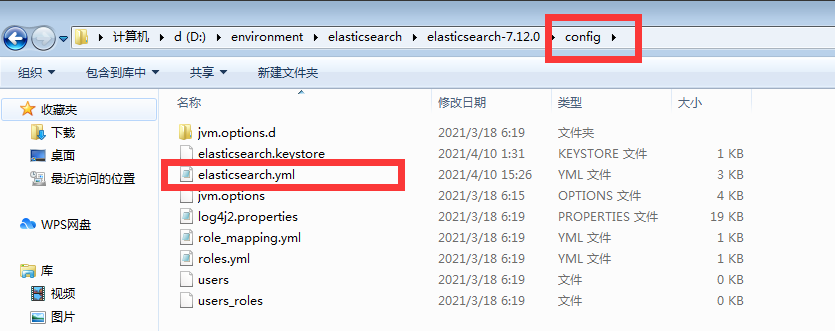
cnpm install //先安装nodejs

npm run start //启动得到文件

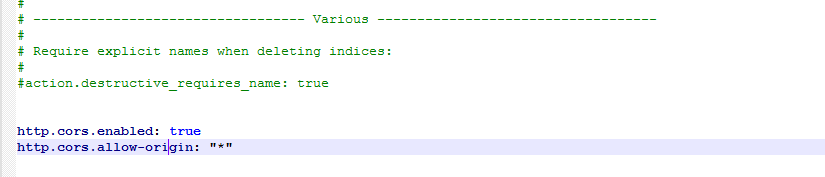
3、出现跨域问题：



1. 到config文件下找到elasticsearch.yml



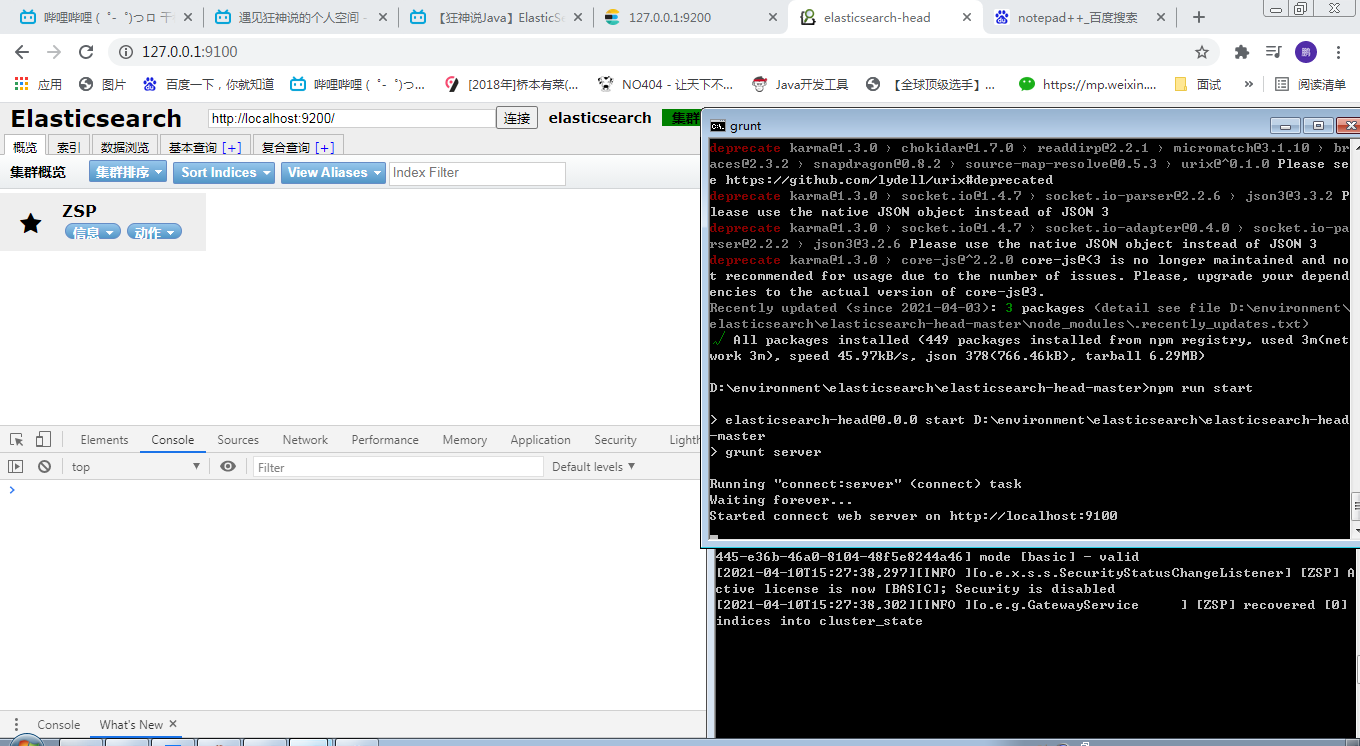
在文件末尾配上：



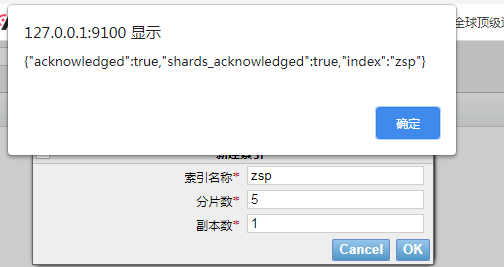
http.cors.enabled: true

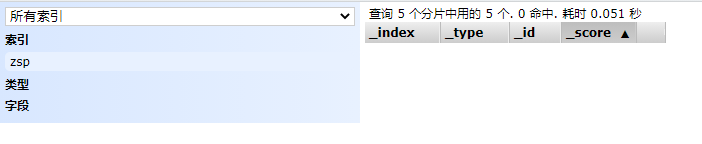
http.cors.allow-origin: "\*"

1. 重启后发现可以连接了



1. 建立个索引





Score是一个权重

你们初学，就把es当做一个数据库!（可以建立索引(库)，文档（库中的数据!） )

这个head我们就把它当做数据展示工具!我们后面所有的查询，Kibana

了解ELK

ELK是Elasticsearch、Logstash、Kibana三大开源框架首字母大写简称。市面上也被成为Elastic Stack。其中Elasticsearch是一个基于Lucene、分布式、通过Restful方式进行交互的近实时搜索平台框架。像类似百度、谷歌这种大数据全文搜索引擎的场景都可以使用Elasticsearch作为底层支持框架，可见Elasticsearch提供的搜索能力确实强大,市面上很多时候我们简称Elasticsearch为es.Logstash是ELK的中央数据流引擎，用于从不同目标（文件/数据存储/MQ）收集的不同格式数据，经过过滤后支持输出到不同目的地(文件/MQ/redis/elasticsearch/kafka等)。Kibana可以将elasticsearch的数据通过友好的页面展示出来，提供实时分析的功能。

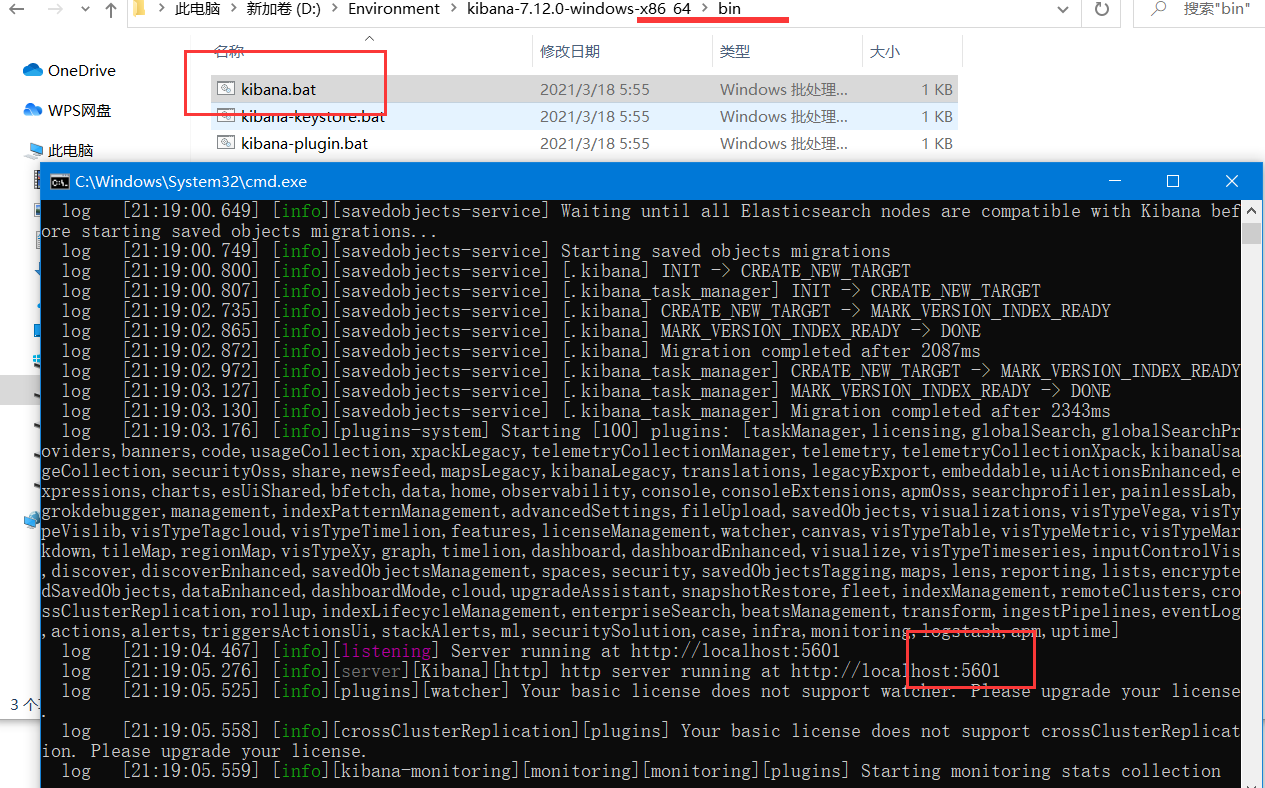


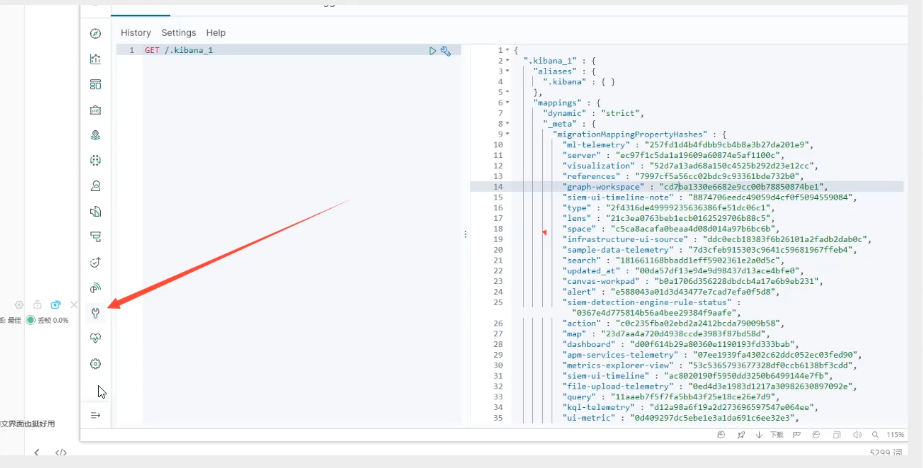
收集清洗数据-->搜索，存储-->Kibana（用kibana进行搜索）

**Kibana：**

Kibana是一个针对Elasticsearch的开源分析及可视化平台，用来搜索、查看交互存储在Elasticsearch索引中的数据。使用Kibana ,可以通过各种图表进行高级数据分析及展示。Kibana让海量数据更容易理解。它操作简单，基于浏览器的用户界面可以快速创建仪表( dashboard ) 实时显示Elasticsearch查询动态。设置Kibana非常简单。无需编码或者额外的基础架构，几分钟内就可以完成Kibana安装并启动Elasticsearch索引监测。

1. 启动测试，启动端口5601

2、开发工具!( Post、curl、head、谷歌浏览器插件测试! )

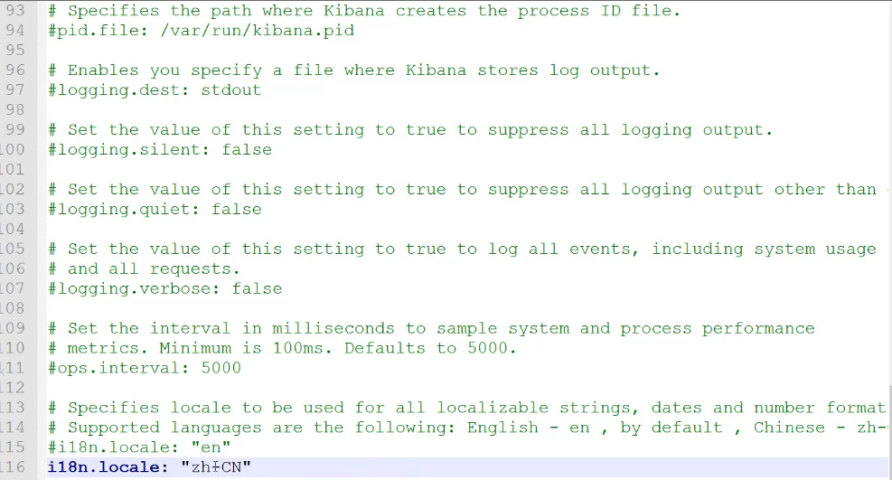


我们之后的所有操作都在这里进行编写!

3、到翻译文件夹下找到i18n



4、中文配置，到config文件夹下进行配置，汉化成功，重启项目即可



**IK分词器：**

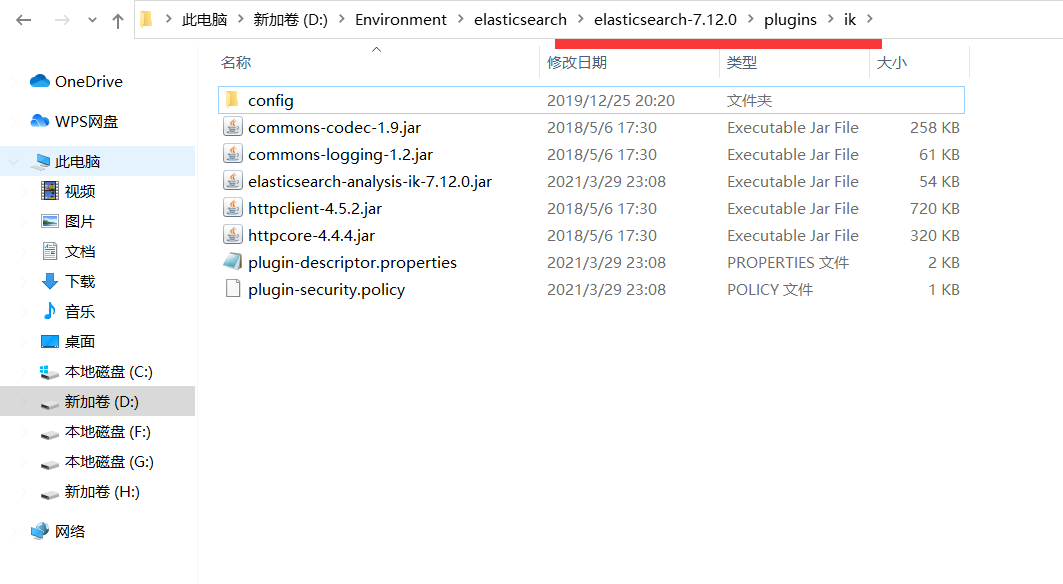
什么是IK分词器?

分词︰即把一段中文或者别的划分成一个个的关键字，我们在搜索时候会把自己的信息进行分词，会把数据库中或者索引库中的数据进行分词，然后进行一个匹配操作，默认的中文分词是将每个字看成一个词，比如“我爱狂神"会被分为"我""爱"狂""神”，这显然是不符合要求的，所以我们需要安装中文分词器ik来解决这个问题。

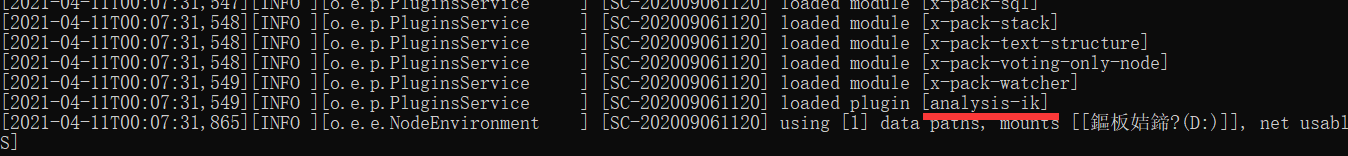
IK提供了两个分词算法: ik\_ smart和ik\_max\_word，其中 ik\_smart为最少切分，ik\_max\_word为最细粒度划分!一会我们测试!

如果要使用中文，建议使用ik分词器!

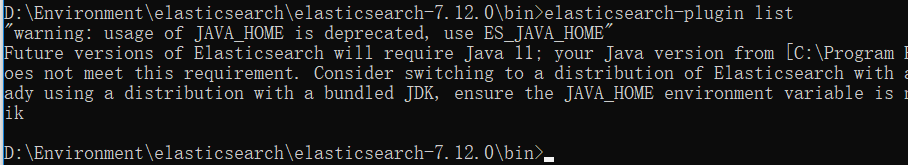
1. 把IK分词器解压到plugins下，这里注意分词器和elastic search一定要相同



1. 重启发现插件已经被安装

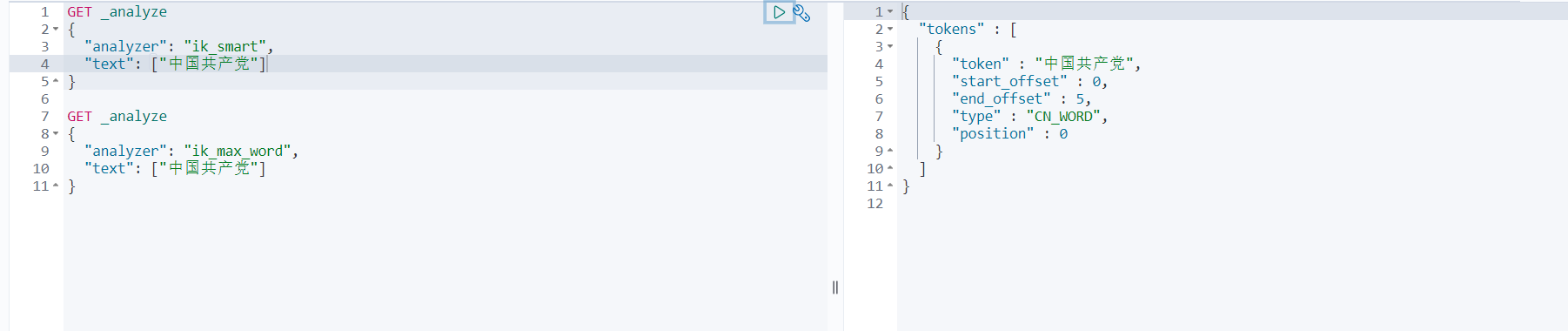


1. 可以使用elasticsearch-plugin list查看安装的插件

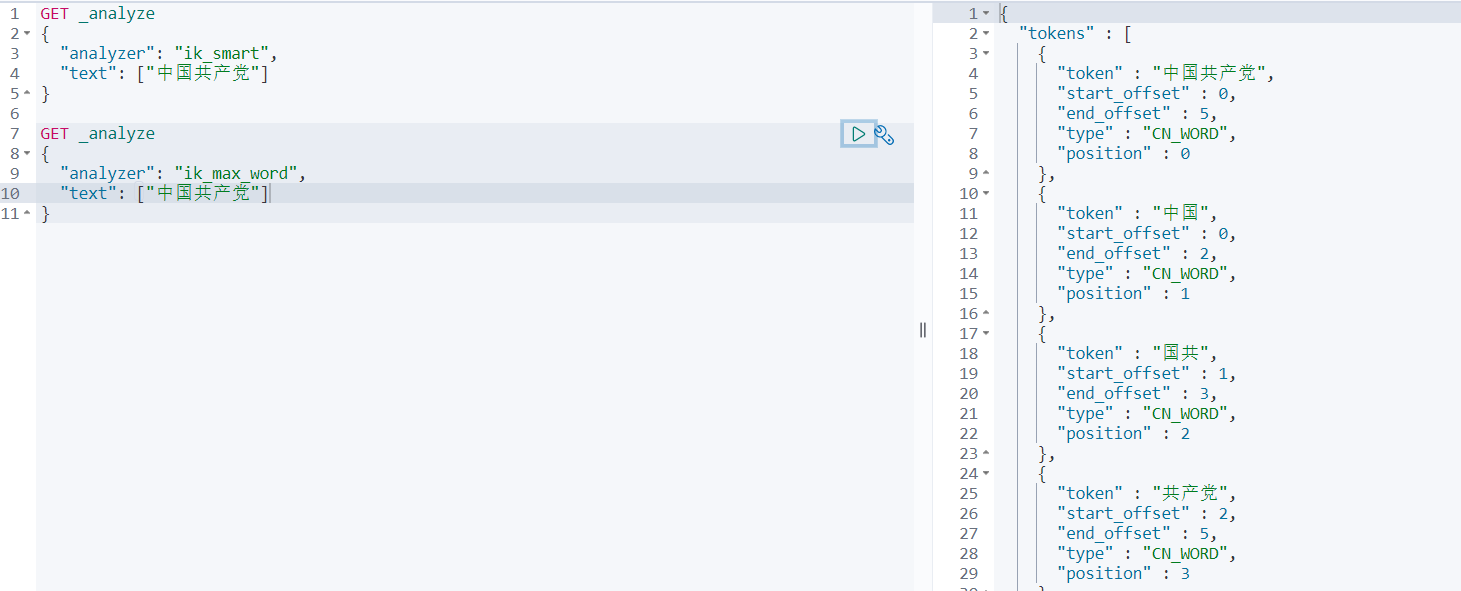


1. 使用Kibana进行IK分词

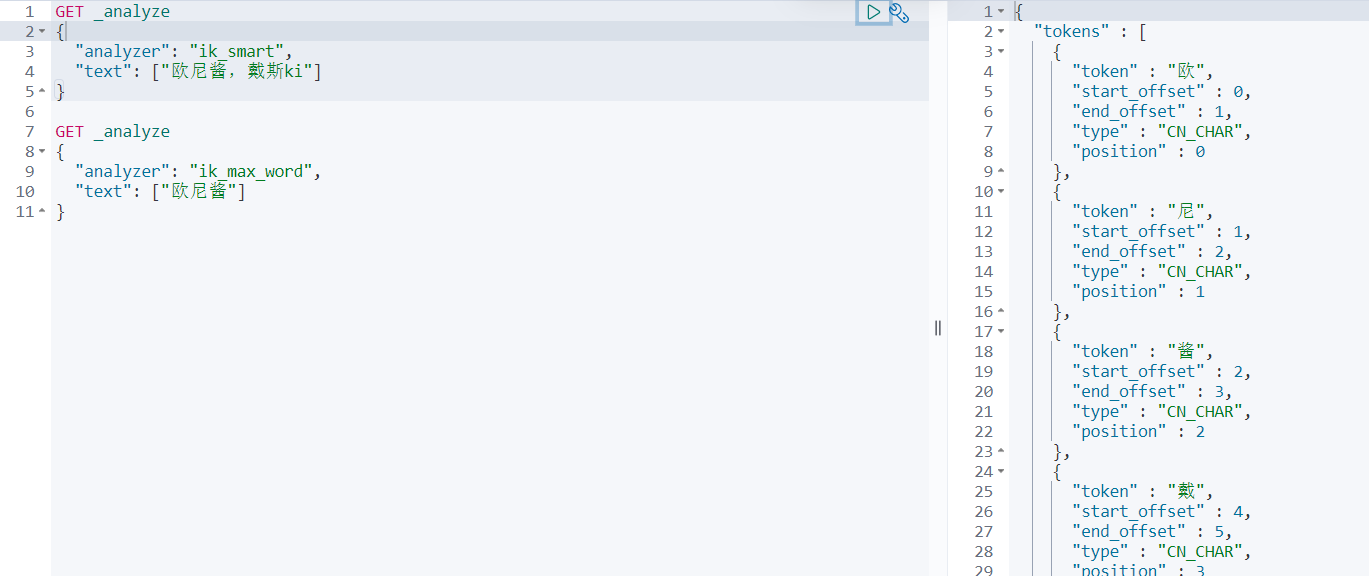
ik\_smart:从词库去找词，最小粒度划分



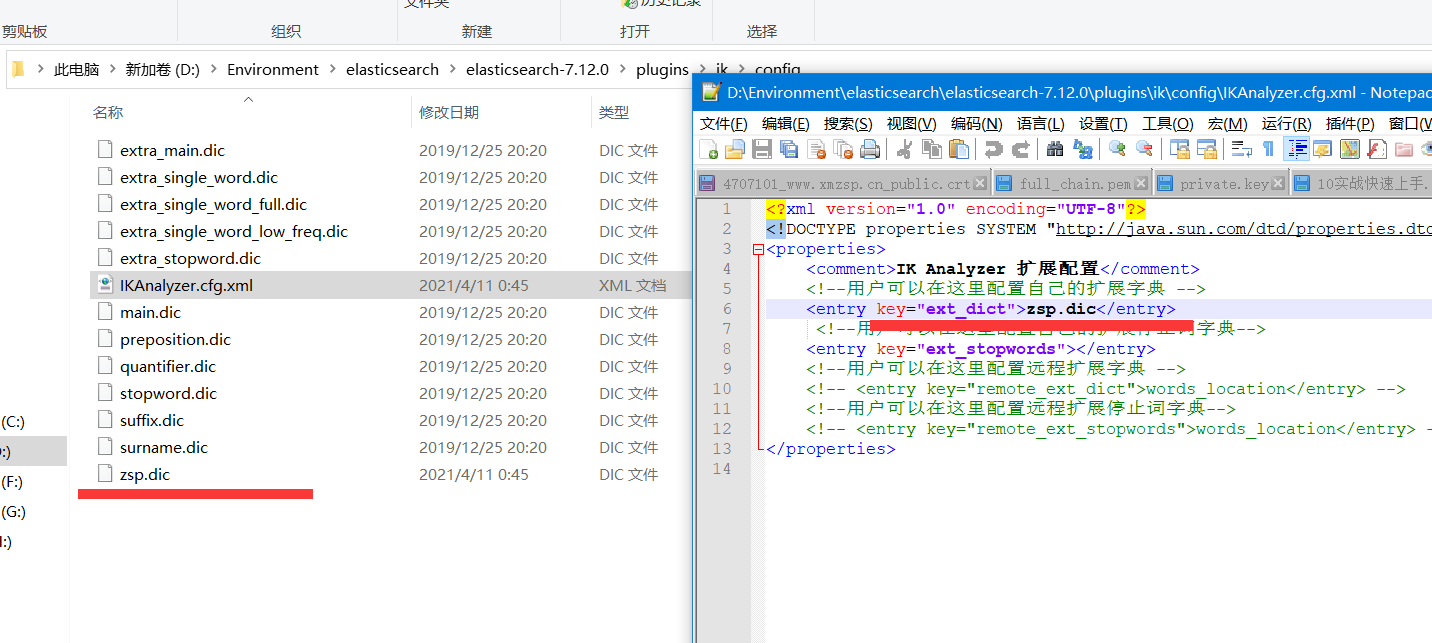
ik\_max\_word:把词按词库，最大拆分



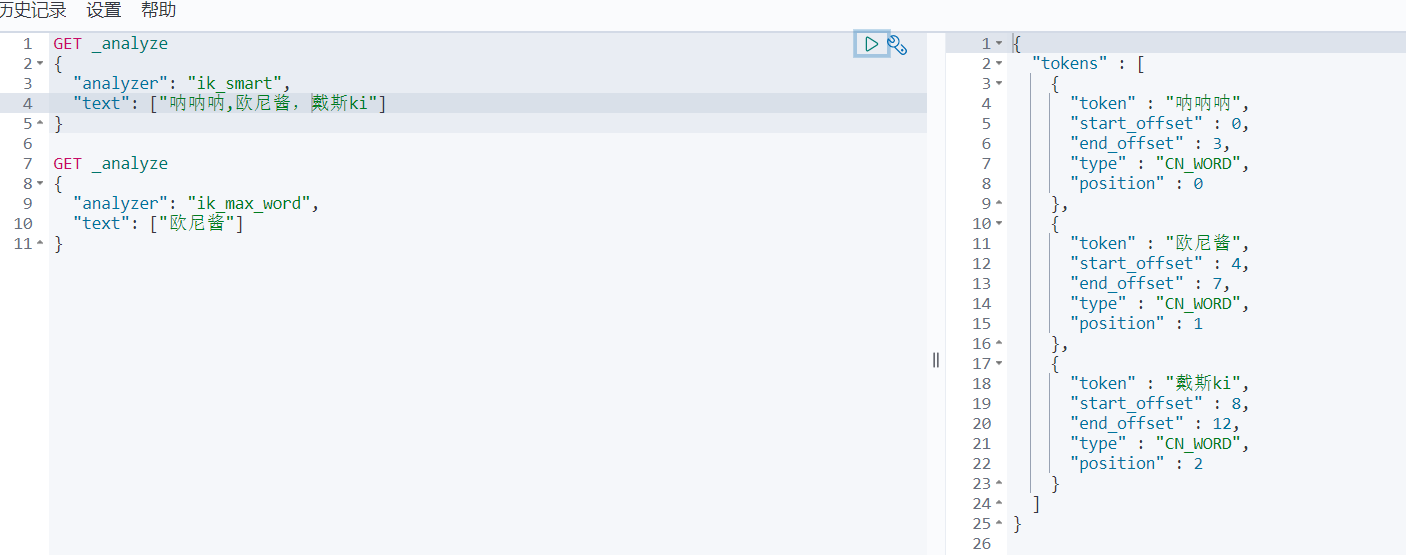
发现问题：欧尼酱和戴斯ki是一个词，不能被分词器拆开



去配置文件配置，配置完毕后重启：



重启完毕后，再次使用分词器：



**Rest风格说明**

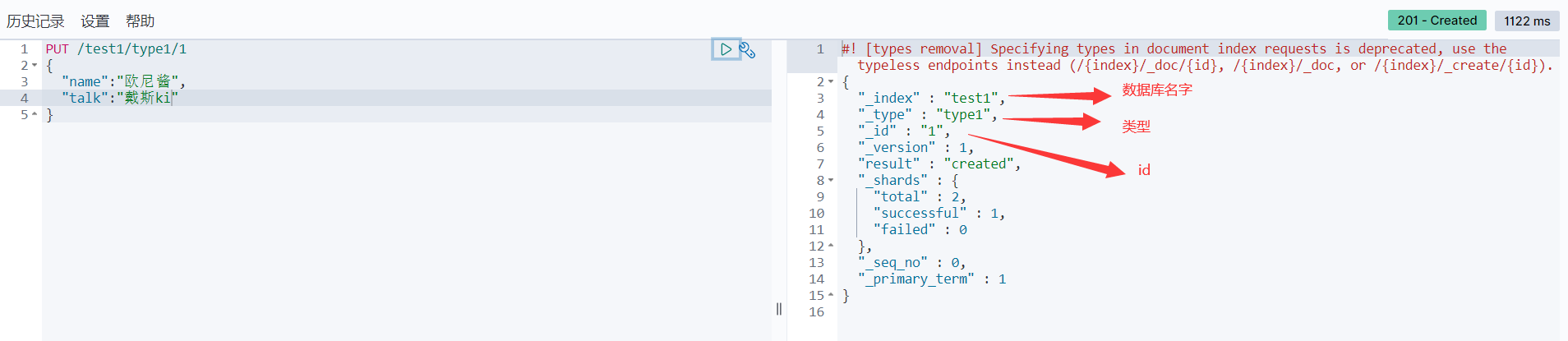
一种软件架构风格，而不是标准，只是提供了一组设计原则和约束条件。它主要用于客户端和服务器交互类的软件。基于这个风格设计的软件可以更简洁，更有层次，更易于实现缓存等机制。

基本Rest命令说明:

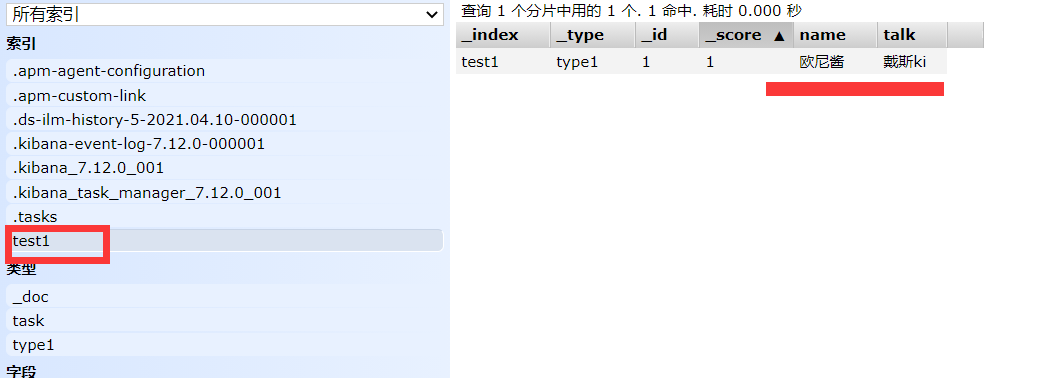


1、基础测试：

PUT /test1/type1/1相当于http://localhost:9200/test1/type1/1。纠正一下：是索引名字为test1



查看一下，的确添加了：



2、那么name这个字段用不用指定类型呢。毕竟我们关系型数据库是需要指定类型的啊!

字符串类型

text 、 keyword

数值类型

long, integer, short, byte, double, float, half float, scaled float·

日期类型

date

te布尔值类型

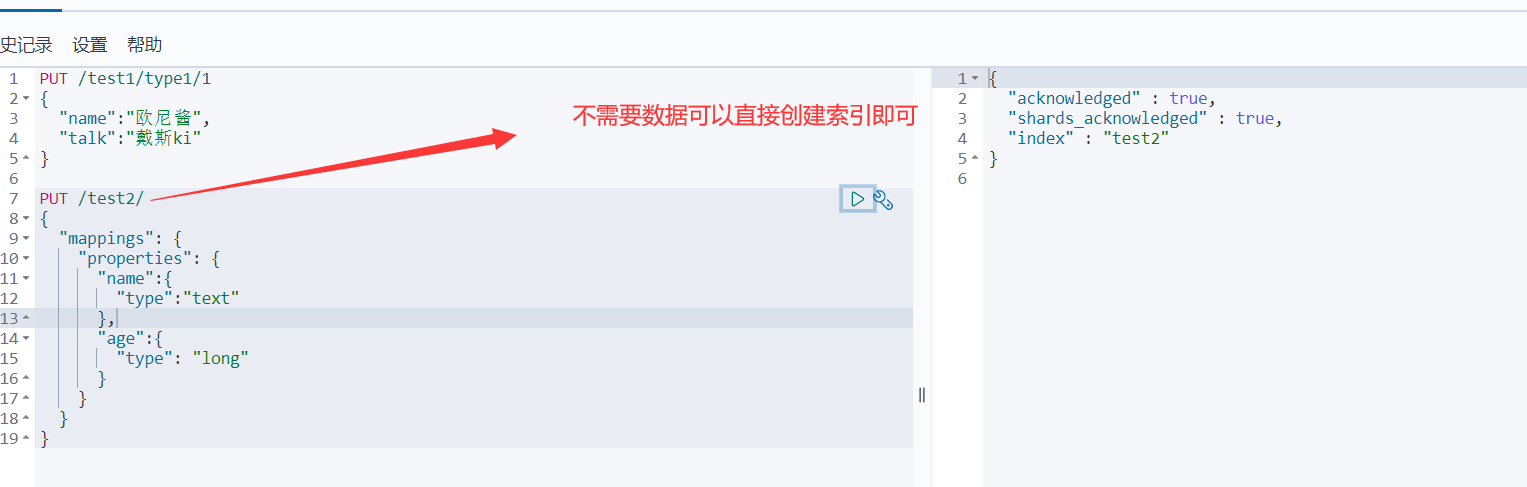
boolean

二进制类型

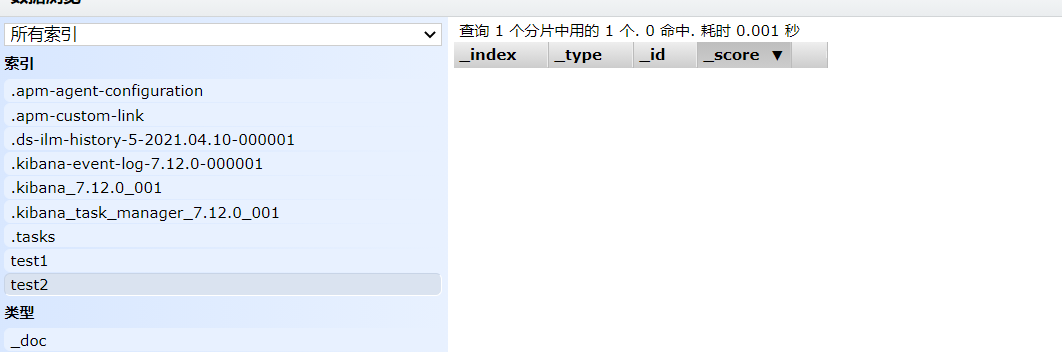
binary

等等…....

3、创建索引规则



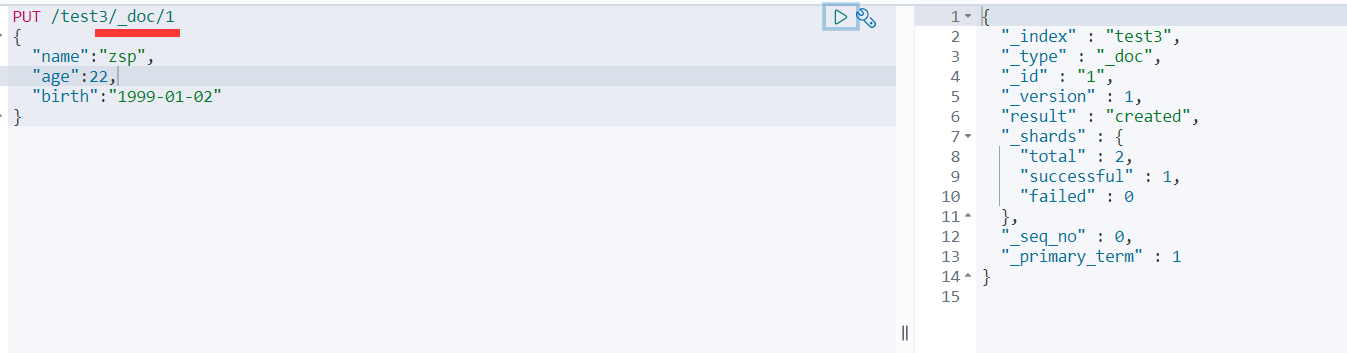
创建成功：



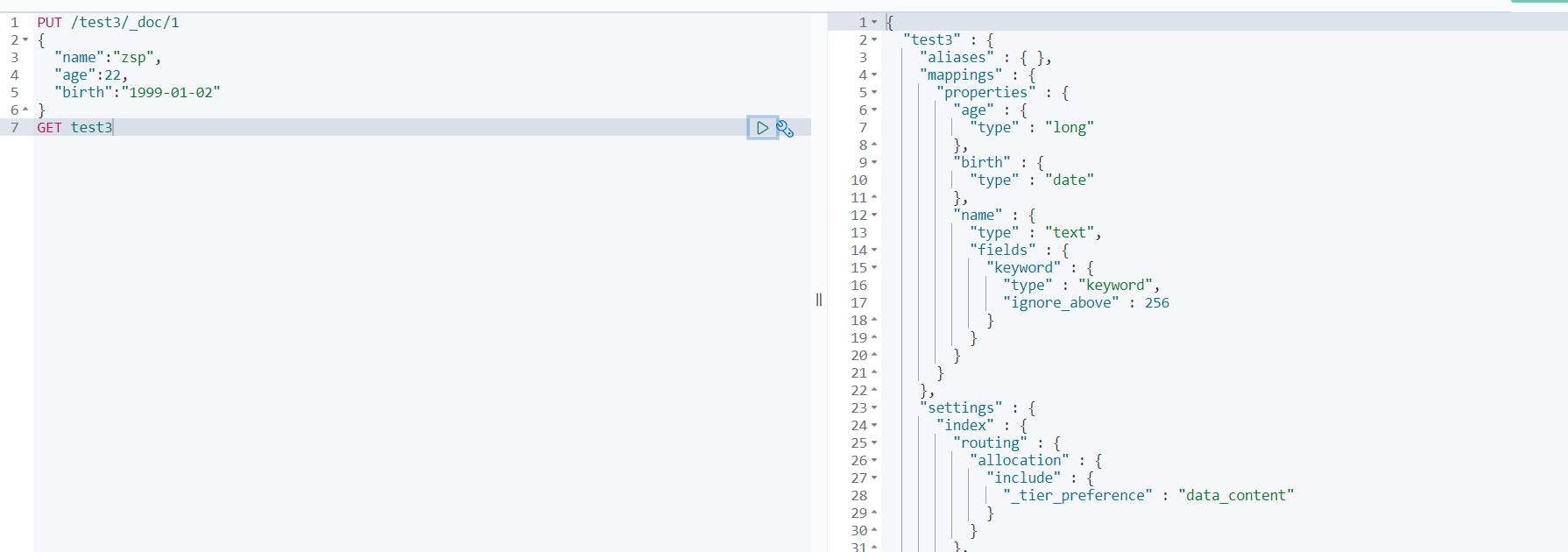
获得这个规则!可以通过GET请求获取具体的信息!



elasticsearch 8.x后的版本索引类型会消失，但是目前还在，换成了\_doc



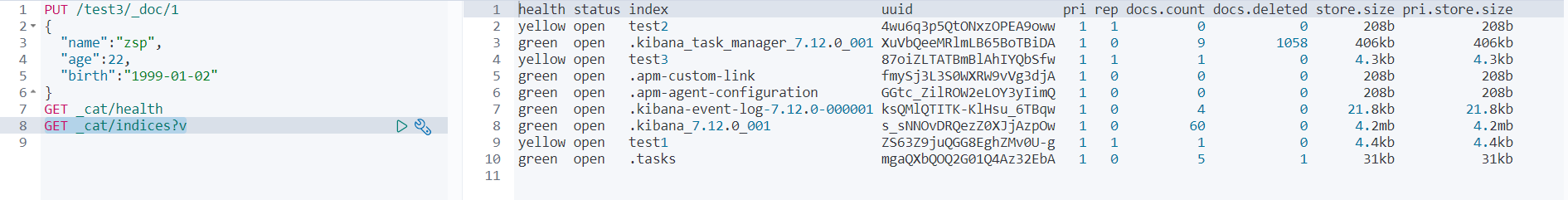
搜索一下，发现系统自动帮忙匹配了类型



还有一些请求：

GET \_cat/health 获取健康情况

GET \_cat/indices?v 获取索引的具体信息



扩展︰通过命令elasticsearch索引情况!通过get \_cat/可以获得es的当前的很多信息!



修改： 提交还是使用PUT 即可!然后覆盖!或者更新

1、使用PUT来进行更新



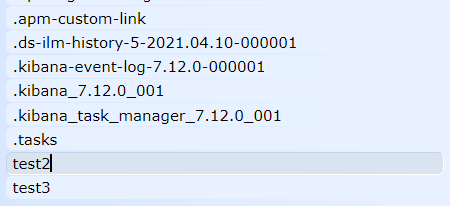
1. 使用update进行更新



删除索引：



删除成功：



**关于文档基本操作：**

1. 添加数据：PUT



1. 获取数据：GET

获取一条索引：

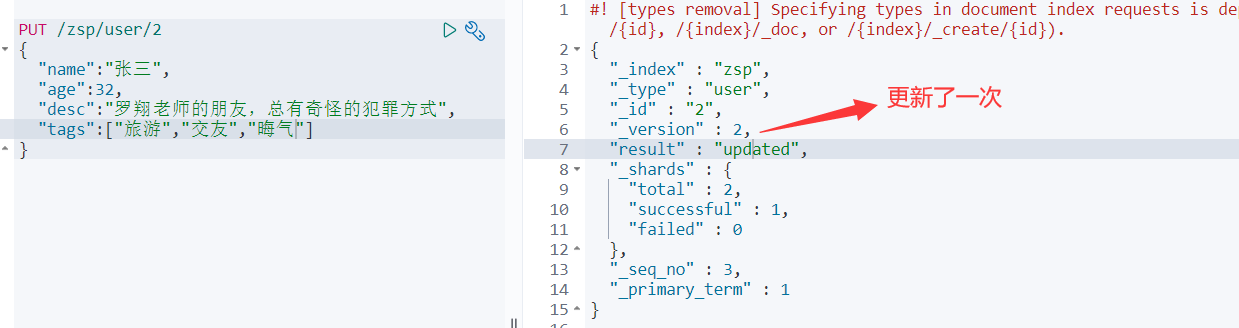


获取全部 /\_search



更新操作：

1. PUT，但数据的每一条都要填写，比较麻烦



1. POST：\_update，推荐使用这种方式



文档简单查询：/\_search?q=name: ，这样查询出来的都是模糊查询，比如sp可以搜出zsp



文档的复杂搜索：也是基于模糊查询，score是权重，权重越高匹配度越强。



结果过滤：加上\_source:[“属性名”,”属性名”]



查询排序：此时的权重无效化



分页查询：from,size



布尔值查询：

must（类似于and）多条件查询：使用bool+must+match，注意match外需要套一层｛｝



should（or）满足其中一个条件即可：



must\_not（not）排除条件：



筛选范围：filter+rang(范围查找，类似于in)



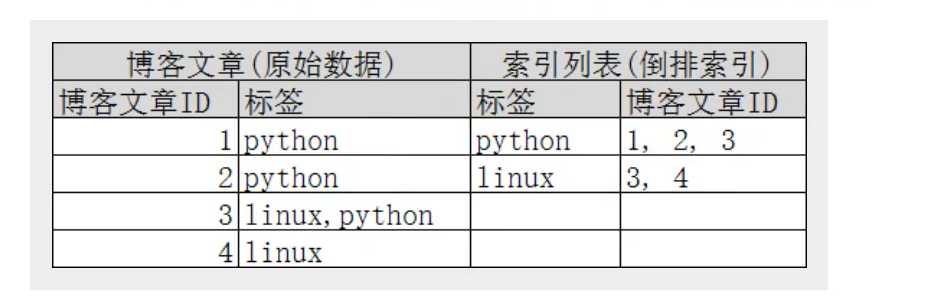
gt：大于 gte：大于等于

lt：小于 lte：小于等于

匹配多个条件查询



精确查询term：term查询是直接通过**倒排索引**指定的词条进程精确查找的!

**关于分词:**

1、term，直接查询精确的，比较快

2、match，会使用分词器解析!(先分析文档，然后在通过分析的文档进行查询!)

**两个类型：text和keyword**

**text：**支持分词，全文检索,支持模糊、精确查询,不支持聚合,排序操作;

**keyword：**不进行分词，直接索引,支持模糊、支持精确匹配，支持聚合、排序操作。



多个值精确匹配查询：



普通高亮：



自定义高亮：



这些其实MySQL也可以做，只是MySQL效率比较低

1、匹配

2、按照条件匹配

3、精确匹配

4、区间范围匹配

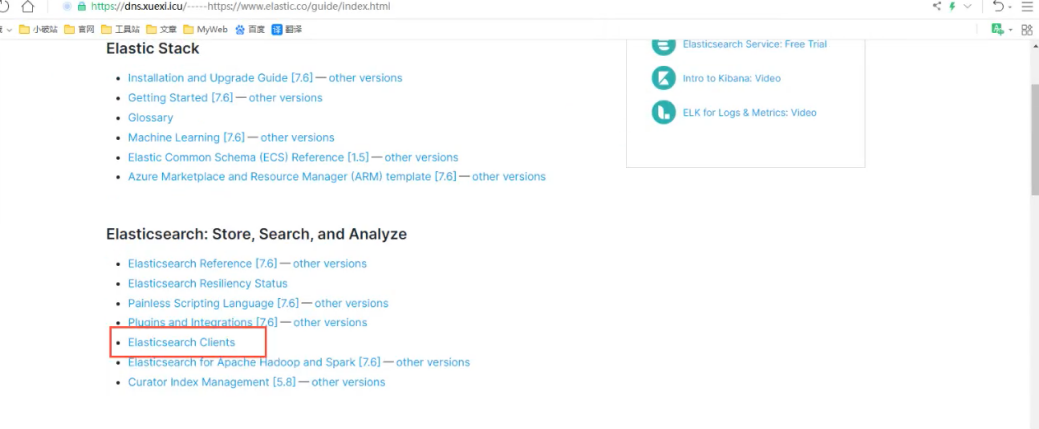
5、匹配字段过滤

6、多条件查询

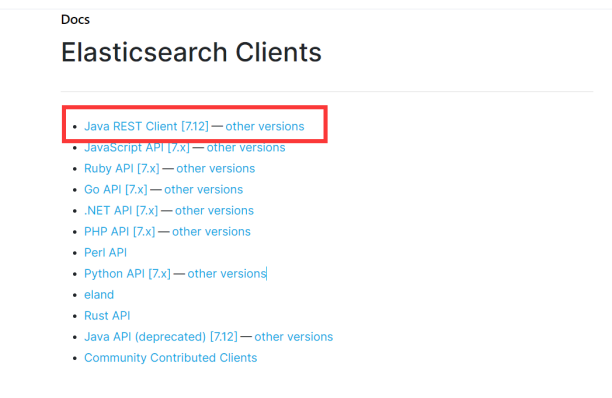
7、高亮查询

**集成SpringBoot：**

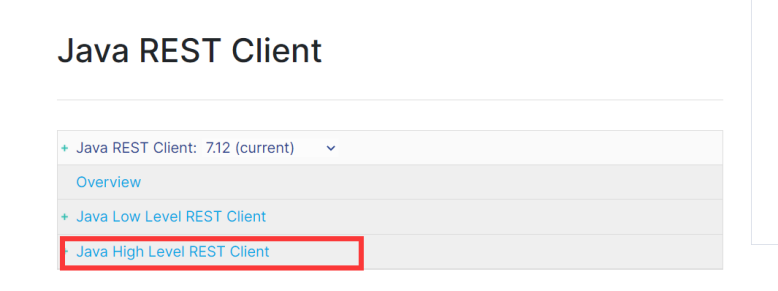
找官方文档：官网的帮助文档



选择使用RESTful：



选择高版本：<https://www.elastic.co/guide/en/elasticsearch/client/java-rest/current/index.html>



1、导入依赖：翻阅文档得知

<repositories>

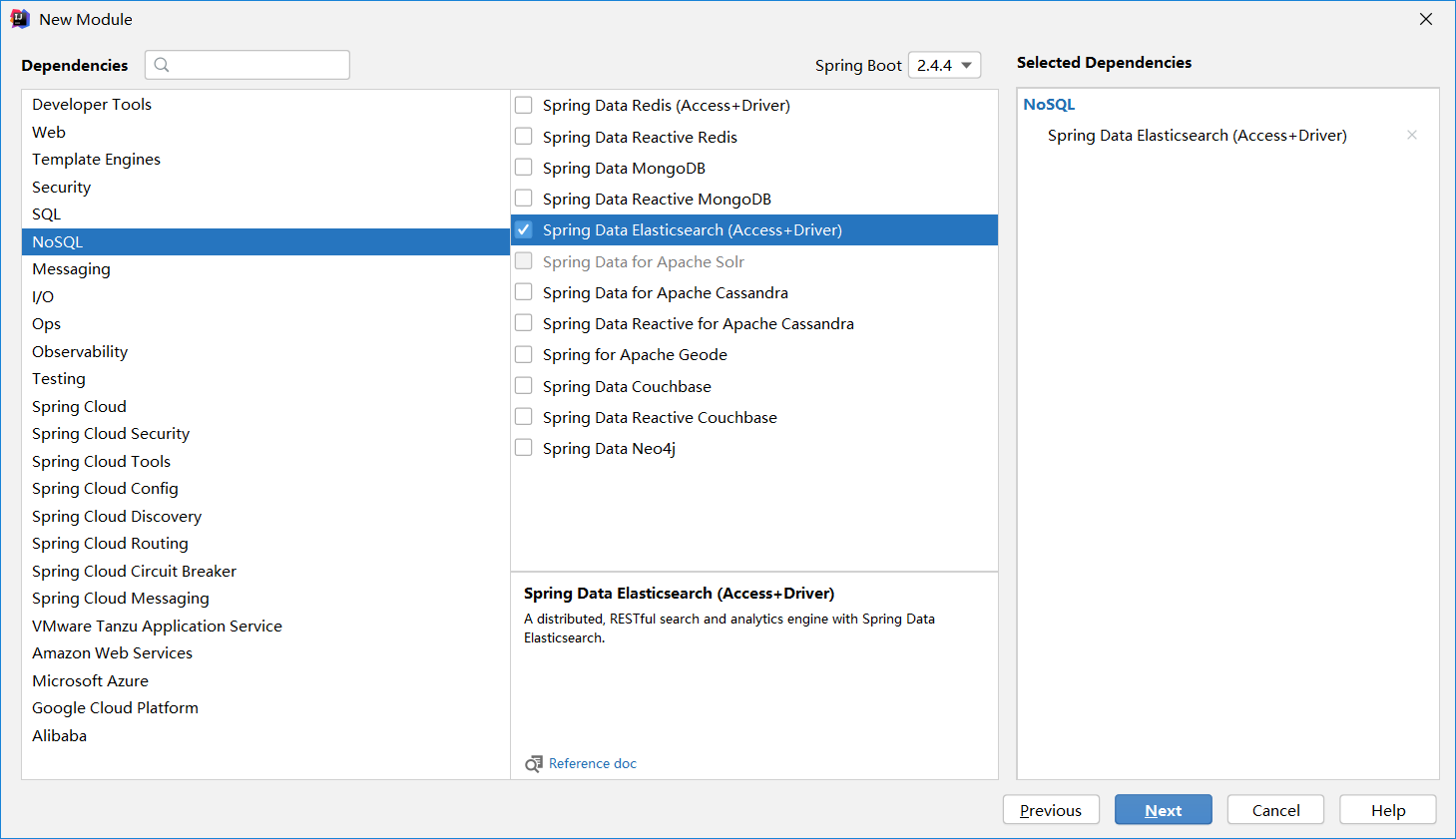
<repository>

<id>es-snapshots</id>

<name>elasticsearch snapshot repo</name>

<url>https://snapshots.elastic.co/maven/</url>

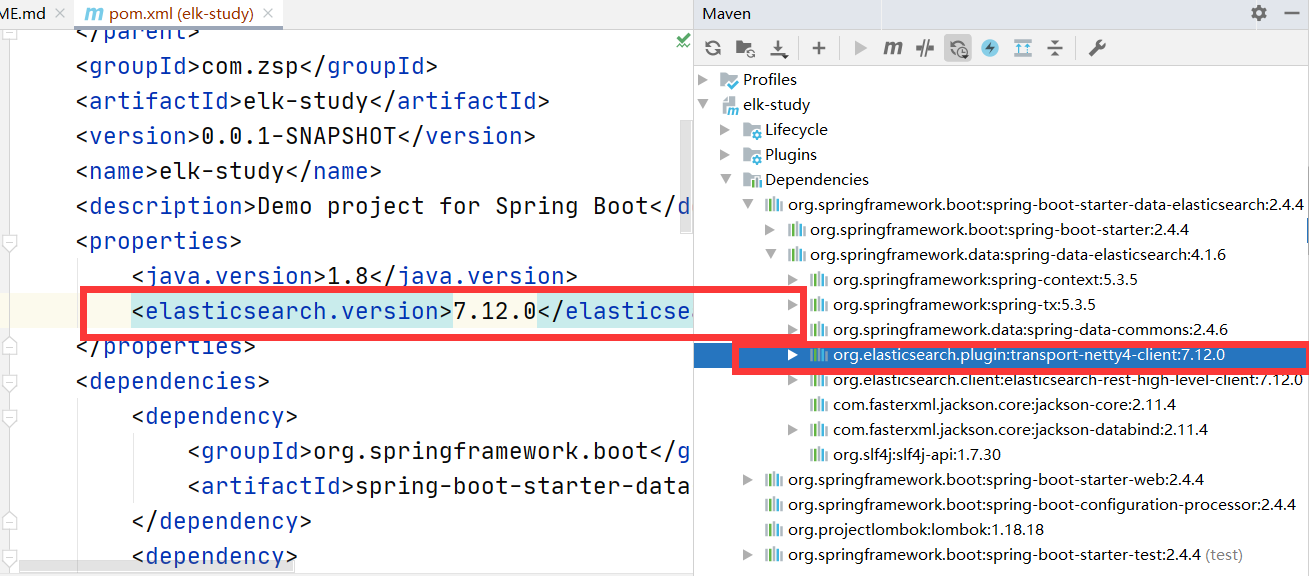
</repository></repositories>



2、找对象



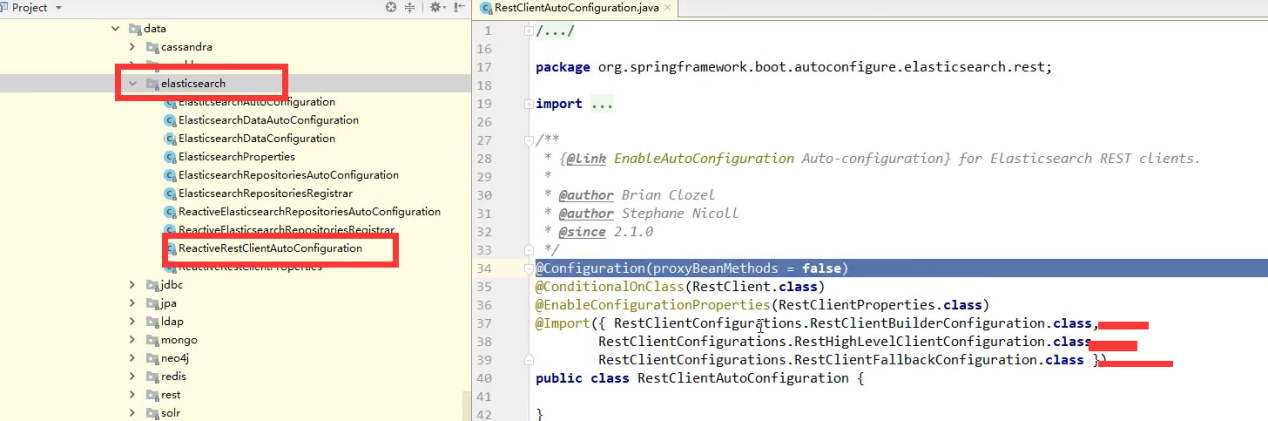
1. 查看库依赖必须和使用的ES版本匹配



1. 建立配置类

@Configuration  
public class ElasticSearchClientConfig {  
 @Bean  
 public RestHighLevelClient restHighLevelClient(){  
 RestHighLevelClient client = new RestHighLevelClient(  
 RestClient.builder(  
 new HttpHost("127.0.0.1", 9200, "http")  
 ));  
  
 return client;  
 }  
}

1. 进入spring-boot-autoconfigure-2.4.4.jar的data里查看elasticsearch源码



找到三个可用对象，分析一下方法得出，虽然这里导入3个类，静态内部类，核心类就一个即高级客户端

**具体API：**

**索引：**

1. 增加索引

@Autowired  
 @Qualifier("restHighLevelClient")  
 RestHighLevelClient client;  
 @Test  
 void testCreate() throws IOException {  
// 创建一个索引  
 CreateIndexRequest request = new CreateIndexRequest("zsp\_index");  
// 将创建参数以及创建的默认请求参数代入  
 CreateIndexResponse createIndexResponse =  
 client.indices().create(request, RequestOptions.DEFAULT);  
// 输出  
 System.out.println(createIndexResponse);  
 }

1. 查找索引

@Test  
void testExit() throws IOException {  
 GetIndexRequest request = new GetIndexRequest("zsp\_index");  
 boolean exists = client.indices().exists(request,RequestOptions.DEFAULT);  
 System.out.println(exists);  
}

1. 删除索引

@Test  
void testDelete() throws IOException {  
 DeleteIndexRequest deleteIndexRequest = new DeleteIndexRequest("zsp\_index");  
 AcknowledgedResponse delete = client.indices().delete(deleteIndexRequest, RequestOptions.DEFAULT);  
 System.out.println(delete.isAcknowledged());  
}

**数据：**

1. 添加数据

void testPut() throws IOException {  
 //创建对象  
 for (int i = 4; i <10 ; i++) {  
 User user = new User("zsp"+i, 18+i);  
 //创建请求  
 IndexRequest request = new IndexRequest("zsp\_index");  
 //设置请求参数,id是必须要的，还有放入对象设置json格式也必须要  
 request.id(""+i);  
 request.source(JSON.toJSONString(user), XContentType.JSON);  
 request.timeout("1s");  
 IndexResponse index = client.index(request, RequestOptions.DEFAULT);  
 }  
}

1. 批量添加数据

@Test  
 void testBluk() throws IOException {  
// 批量插入使用  
 BulkRequest bulkRequest = new BulkRequest();  
 ArrayList<User> userList = new ArrayList<>();  
 userList.add(new User("psz",20));  
 userList.add(new User("gg",21));  
 userList.add(new User("ps",22));  
 userList.add(new User("pz",23));  
 for (User user : userList) {  
 bulkRequest.add(new IndexRequest("zsp\_index")  
// .id() //这里可以填入id,不指定就是随机  
 .source(JSON.toJSONString(user),XContentType.JSON));  
 }  
 client.bulk(bulkRequest,RequestOptions.DEFAULT);  
 }

1. 删除数据

@Test  
void testDelRequest() throws IOException {  
 DeleteRequest request = new DeleteRequest("zsp\_index");  
 request.id("9");  
 DeleteResponse delete = client.delete(request, RequestOptions.DEFAULT);  
 System.out.println(delete.status());  
}

1. 更新数据

@Test  
void testUpdate() throws IOException {  
 UpdateRequest updateRequest = new UpdateRequest();  
 User user = new User("ppp", 18);  
 updateRequest.index("zsp\_index").id("1").doc(JSON.toJSONString(user),XContentType.JSON);  
 client.update(updateRequest,RequestOptions.DEFAULT);  
}

1. 批量查询

@Test  
 void testSearch() throws IOException {  
// 1、建立请求  
 SearchRequest searchRequest = new SearchRequest("zsp\_index");  
// 2、建立搜索对象  
 SearchSourceBuilder sourceBuilder = new SearchSourceBuilder();  
// 3、简历搜索的工具类，使用工具类规定查找方法  
 TermQueryBuilder termQueryBuilder = QueryBuilders.termQuery("name","zsp");  
// QueryBuilders.matchAllQuery();//查询所有  
// sourceBuilder.from() //从xx开始 分页  
// sourceBuilder.size() //长度多少 分页  
// sourceBuilder.highlighter();//高亮  
// 4、将搜索工具丢入对象  
 sourceBuilder.query(termQueryBuilder);  
// 5、将搜索方法丢入请求  
 searchRequest.source(sourceBuilder);  
// 6、将搜索请求发送给ES  
 SearchResponse searchResponse = client.search(searchRequest, RequestOptions.DEFAULT);  
 for (SearchHit hit : searchResponse.getHits()) {  
 System.out.println(hit );  
 }  
 }

**项目实战**

**导入依赖**

<dependencies>  
<!-- 解析网页jsoup-->  
 <dependency>  
 <groupId>org.jsoup</groupId>  
 <artifactId>jsoup</artifactId>  
 <version>1.10.2</version>  
 </dependency>  
<!-- json工具-->  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>fastjson</artifactId>  
 <version>1.2.62</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-elasticsearch</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-thymeleaf</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-devtools</artifactId>  
 <scope>runtime</scope>  
 <optional>true</optional>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-configuration-processor</artifactId>  
 <optional>true</optional>  
 </dependency>  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <optional>true</optional>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>

**爬虫网页（主要用jsoup，但是爬取不了视频和音频。Tika可以）**