

狂神聊ElasticSearch

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
- 8、SpringBoot 集成 ElasticSearch（从原理分析！）
- 9、爬虫爬取数据！
- 10、实战，模拟全文检索！

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！）

聊聊Doug Cutting

本故事内容来自公众号：鲜枣课堂

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**。



左为Doug Cutting，右为Lucene的LOGO

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**。因为好用而且**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。



Apache软件基金会，搞IT的应该都认识

2004年，Doug Cutting再接再厉，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流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。



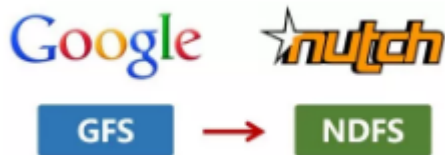
Google搜索栏

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。

大数据就两个问题：存储 + 计算！

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 (Nutch Distributed 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。



这里要补充说明一下雅虎招安Doug的背景：2004年之前，作为互联网开拓者的雅虎，是使用Google搜索引擎作为自家搜索服务的。在2004年开始，雅虎放弃了Google，开始自己研发搜索引擎。所以。。。

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，Hadoop Distributed 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。

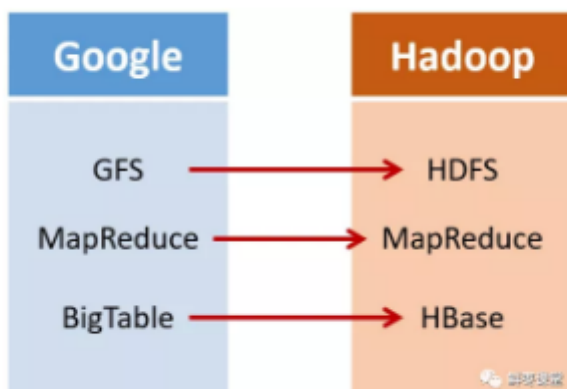
这次，它们介绍了自己的**BigTable**。这是一种分布式数据存储系统，一种用来处理海量数据的非关系型数据库。

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，反正就是紧跟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。

[回到主题](#)

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！）

我的讲课风格：学习更多的是培养大家的学习兴趣！

教学风格：开源、免费的、授人以渔！

只要学不死，就往死里学！

ElasticSearch概述

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（大数据时代）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。

据国际权威的数据库产品评测机构DB Engines的统计，在2016年1月，ElasticSearch已超过Solr等，**成为排名第一的搜索引擎类应用**。

[历史](#)

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.....

现在我们就知道了 elasticsearch 重要性！

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（权重，百度！）
- 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、BI系统，商业智能，Business Intelligence。比如说有个大型商场集团，BI，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

架构选择！

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：

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-as-you-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Java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Java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序接口，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。

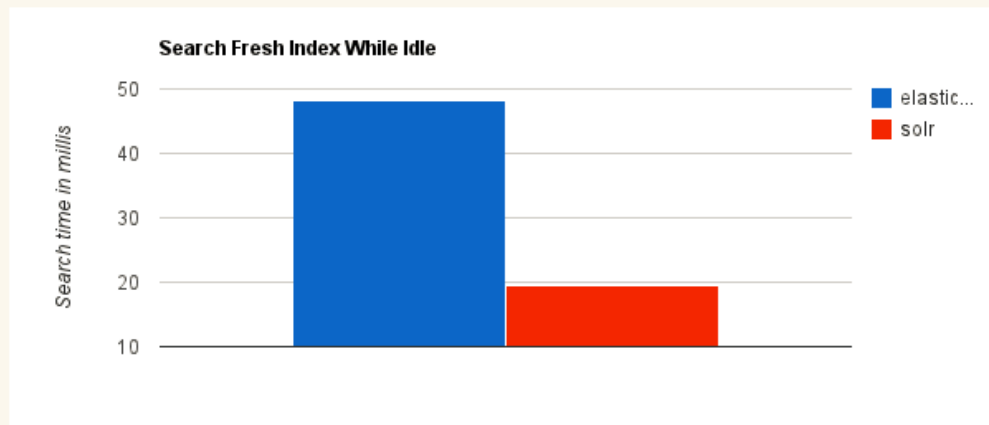
Lucene是一个全文检索引擎的架构。那什么是全文搜索引擎？

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、Fast/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。

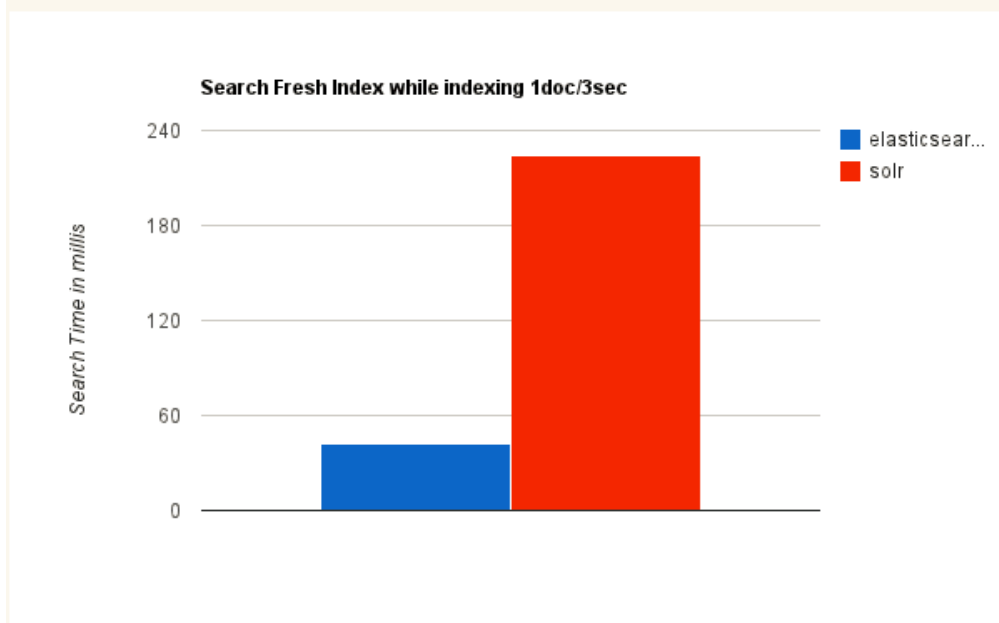
从搜索结果来源的角度，全文搜索引擎又可细分为两种，一种是拥有自己的检索程序（Indexer），俗称“蜘蛛”（Spider）程序或“机器人”（Robot）程序，并自建网页数据库，搜索结果直接从自身的数据库中调用，如上面提到的7家引擎；另一种则是租用其他引擎的数据库，并按自定的格式排列搜索结果，如Lycos引擎。

Elasticsearch和Solr比较

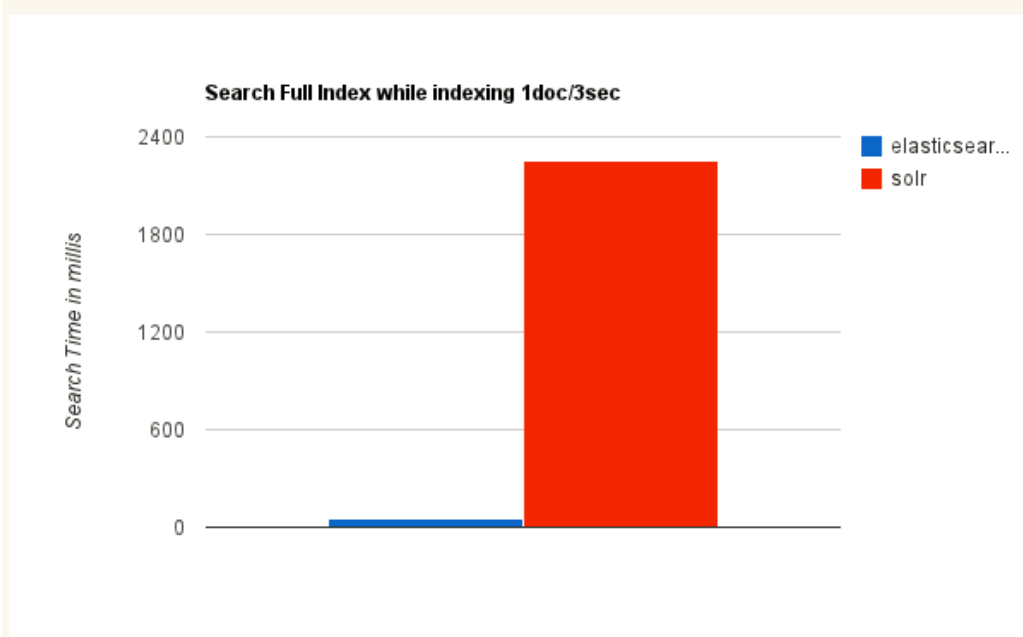
当单纯的对已有数据进行搜索时，Solr更快。



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!



ElasticSearch 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 本身更侧重于核心功能，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。
- 6、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！）

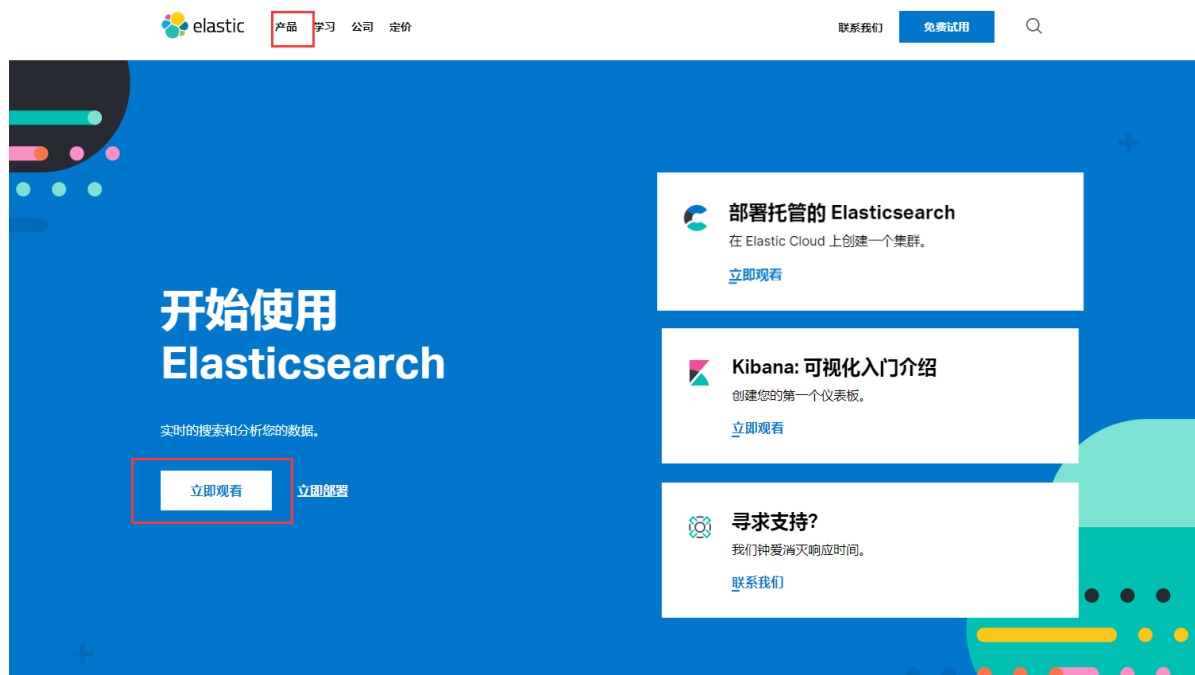
ElasticSearch安装

声明：JDK1.8，最低要求！ElasticSearch 客户端，界面工具！

Java开发，ElasticSearch 的版本和我们之后对应的 Java 的核心jar包！版本对应！JDK 环境是正常！

下载

官网：<https://www.elastic.co/>



- 通过CRUD REST API 添加，更新，检索和删除数据
- 基本的文本分析，包括标记和过滤
- 基本搜索查询
- 聚合：Elasticsearch 的面向和分析的主功能

其他资源：

- [Elasticsearch 文档](#)
- [下载Elasticsearch](#)
- [7.x 发布的博客](#)

什么是 Elasticsearch?

Elasticsearch 是一个开源的分布式 RESTful 搜索和分析引擎，能够解决越来越多不同的应用场景。此外，还可以[免费试用14天 阿里云 Elasticsearch 服务](#)或[腾讯云Elasticsearch服务](#)。

Elasticsearch

功能 定价 合规性 云状态

Download Elasticsearch

Want it hosted? Deploy on Elastic Cloud. [Get Started »](#)

Version: 7.6.2

Release date: April 01, 2020

License: [Elastic License](#)

Downloads:

[WINDOWS](#) sha512
 [MACOS](#) sha512
 [DEB](#) sha512
 [MSI \(BETA\)](#) sha512

[LINUX](#) sha512
 [RPM](#) sha512

Package Managers:

Install with [yum, dnf, or zypper](#)

Install with [apt-get](#)

Install with [homebrew](#)

Containers: Run with [Docker](#)

Notes:

Running on Kubernetes? Try [Elastic Cloud on Kubernetes](#) or the Elasticsearch [Helm Chart \(beta\)](#).

This default distribution is governed by the Elastic License, and includes the [full set of free features](#).

View the detailed release notes [here](#).

下载地址：<https://www.elastic.co/cn/downloads/elasticsearch>

官网下载巨慢，翻墙，网盘中下载即可！

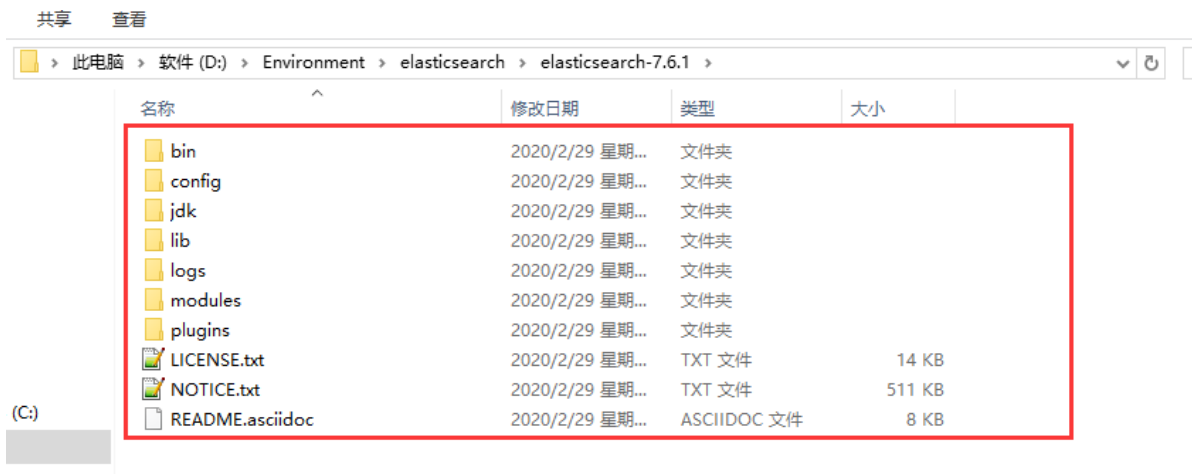
我们学习的话 Window 和 Linux 都可以学习！

我们这里现在Window下学习！

ELK 三剑客，解压即用！（web项目！前端环境！）

window 下安装！

1、解压就可以使用了！



2、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！

3、启动，访问9200；

```
x_open_jobs=20} elect leader, BECOME_MASTER_TASK, FINISH_ELECTION ], term: 1, version: 1, delta: master node changed {pre
s [], current [{KUANGSHEN} {u03snfFsRbONtulpFmpo7w} {e4Mbf_J-QsaBdRrIiVYEQA} {127.0.0.1} {127.0.0.1:9300} {dilm} {ml.machine_memory=8529625088, xpack.installed=true, ml.max_open_jobs=20}]]
20-04-02T21:16:03.137 [INFO] [o.e.c.c.CoordinationState] [KUANGSHEN] cluster UUID set to [Cpbykv0VTwyqyWaa9ozBhw]
20-04-02T21:16:03.185 [INFO] [o.e.c.s.ClusterApplierService] [KUANGSHEN] master node changed {previous [], current [{KUANGSHEN} {u03snfFsRbONtulpFmpo7w} {e4Mbf_J-QsaBdRrIiVYEQA} {127.0.0.1} {127.0.0.1:9300} {dilm} {ml.machine_memory=8529625088, xpack.installed=true, ml.max_open_jobs=20}]], term: 1, version: 1, reason: Publication(term=1, version=1)}
20-04-02T21:16:03.277 [INFO] [o.e.g.GatewayService] [KUANGSHEN] recovered [0] indices into cluster state
20-04-02T21:16:03.370 [INFO] [o.e.h.AbstractHttpServerTransport] [KUANGSHEN] publish_address {127.0.0.1:9200}, bound_address {127.0.0.1:9200}, {[:1]:9200}
20-04-02T21:16:03.372 [INFO] [o.e.n.Node] [KUANGSHEN] started
```

4、访问测试！



安装可视化界面 es head的插件

没有前端基础的，先去看我的Vue，把基本的环境安装完毕！

1、下载地址：<https://github.com/mobz/elasticsearch-head/>

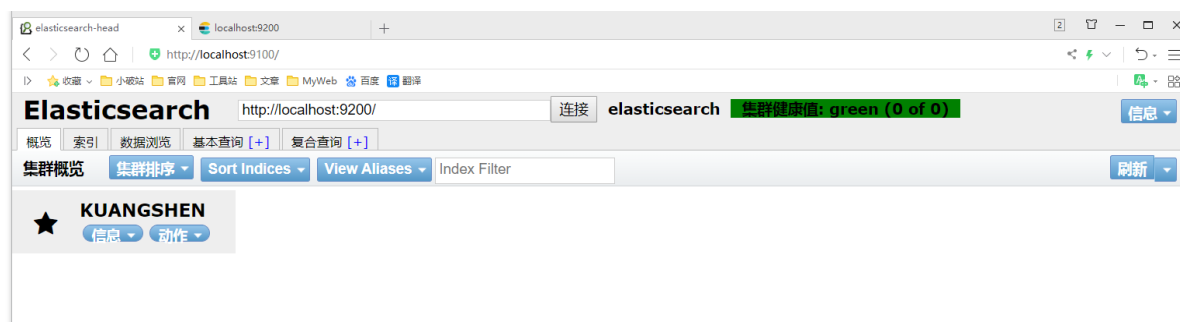
2、启动

```
npm install
npm run start
```

3、连接测试发现，存在跨域问题：配置es

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

4、重启es服务器，然后再次连接



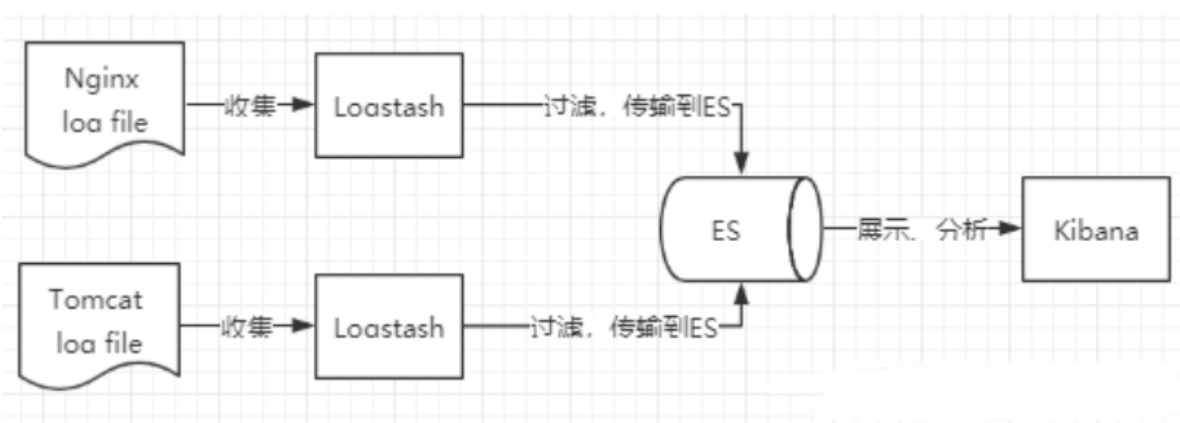
你们初学，就把es当做一个数据库！（可以建立索引（库），文档（库中的数据！））

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。

市面上很多开发只要提到ELK能够一致说出它是一个日志分析架构技术栈总称，但实际上ELK不仅仅适用于日志分析，它还可以支持其它任何数据分析和收集的场景，日志分析和收集只是更具有代表性。并非唯一性。



安装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盘（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。

官网：<https://www.elastic.co/cn/kibana>

Kibana 版本要和 Es 一致！



Download Kibana

Want it hosted? Deploy on Elastic Cloud. [Get Started »](#)

Version: 7.6.2

Release date: April 01, 2020

License: [Elastic License](#)

Downloads: [WINDOWS](#) [sha](#) [asc](#) [MAC](#) [sha](#) [asc](#)
[LINUX 64-BIT](#) [sha](#) [asc](#) [RPM 64-BIT](#) [sha](#) [asc](#)
[DEB 64-BIT](#) [sha](#) [asc](#)

Package Managers: Install with [yum](#), [dnf](#), or [zypper](#)
Install with [apt-get](#)
Install with [homebrew](#)

Containers: Run with [Docker](#)

Notes: Running on Kubernetes? Try [Elastic Cloud on Kubernetes](#) (alpha) or the Kibana [Helm Chart](#) (beta).

下载完毕后，解压也需要一些时间！是一个标准的工程！

好处：ELK 基本上都是拆箱即用！

启动测试：

1、解压后端的目录

此电脑 > 软件 (D:) > Environment > elasticsearch > kibana-7.6.1-windows-x86_64 >

名称	修改日期	类型	大小
bin	2020/2/29 星期...	文件夹	
built_assets	2020/2/29 星期...	文件夹	
config	2020/2/29 星期...	文件夹	
data	2020/2/29 星期...	文件夹	
node	2020/2/29 星期...	文件夹	
node_modules	2020/2/29 星期...	文件夹	
optimize	2020/2/29 星期...	文件夹	
plugins	2020/2/29 星期...	文件夹	
src	2020/2/29 星期...	文件夹	
webpackShims	2020/2/29 星期...	文件夹	
x-pack	2020/2/29 星期...	文件夹	
.i18nrc.json	2020/2/29 星期...	JSON 文件	3 KB
LICENSE.txt	2020/2/29 星期...	TXT 文件	14 KB
NOTICE.txt	2020/2/29 星期...	TXT 文件	1,688 KB
package.json	2020/2/29 星期...	JSON 文件	1 KB
README.txt	2020/2/29 星期...	TXT 文件	4 KB

2、启动

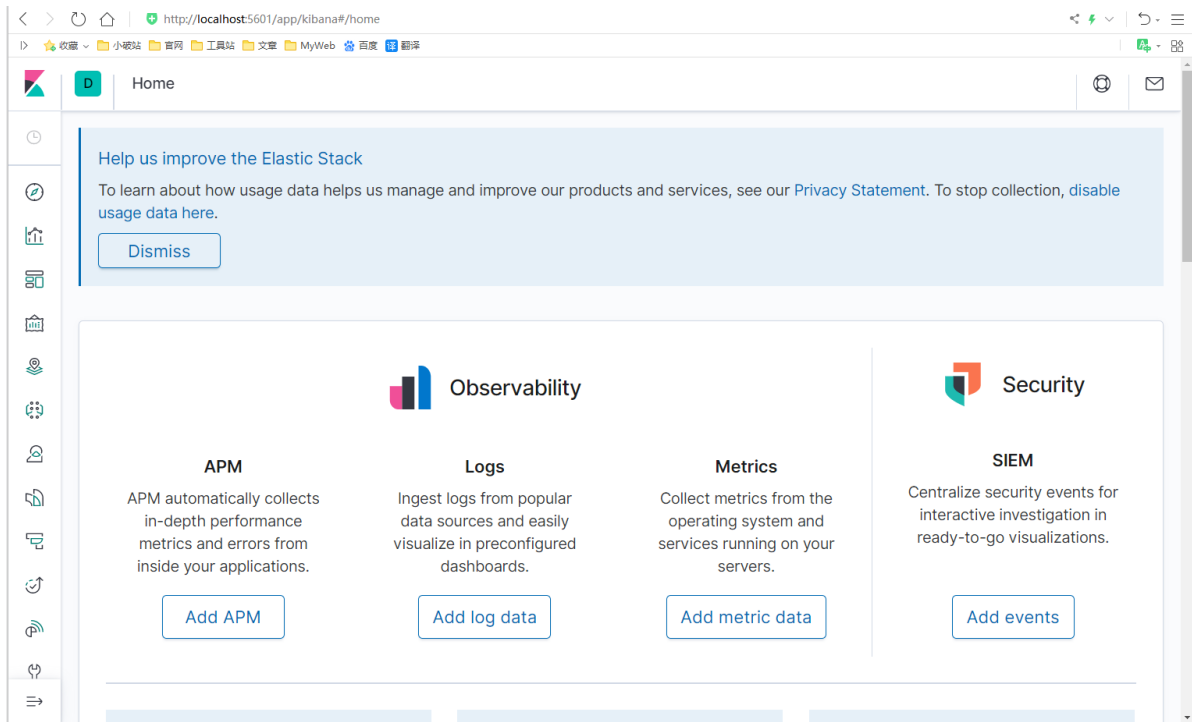
```

:40:33.035] [info][status][plugin:remote_clusters@7.6.1] Status changed from uninitialized to green - Read
:40:33.037] [info][status][plugin:cross_cluster_replication@7.6.1] Status changed from uninitialized to gr
:40:33.050] [info][status][plugin:upgrade_assistant@7.6.1] Status changed from uninitialized to green - Re
:40:33.074] [info][status][plugin:uptime@7.6.1] Status changed from uninitialized to green - Ready
:40:33.078] [info][status][plugin:oss_telemetry@7.6.1] Status changed from uninitialized to green - Ready
:40:33.081] [info][status][plugin:file_upload@7.6.1] Status changed from uninitialized to green - Ready
:40:33.082] [info][status][plugin:data@7.6.1] Status changed from uninitialized to green - Ready
:40:33.085] [info][status][plugin:lens@7.6.1] Status changed from uninitialized to green - Ready
:40:33.099] [info][status][plugin:snapshot_restore@7.6.1] Status changed from uninitialized to green - Rea
:40:33.107] [info][status][plugin:input_control_vis@7.6.1] Status changed from uninitialized to green - Re
:40:33.113] [info][status][plugin:kibana_react@7.6.1] Status changed from uninitialized to green - Ready
:40:33.116] [info][status][plugin:management@7.6.1] Status changed from uninitialized to green - Ready
:40:33.117] [info][status][plugin:navigation@7.6.1] Status changed from uninitialized to green - Ready
:40:33.118] [info][status][plugin:region_map@7.6.1] Status changed from uninitialized to green - Ready
:40:33.128] [info][status][plugin:telemetry@7.6.1] Status changed from uninitialized to green - Ready
:40:33.130] [info][status][plugin:ui_metric@7.6.1] Status changed from uninitialized to green - Ready
:40:33.217] [info][status][plugin:timelion@7.6.1] Status changed from uninitialized to green - Ready
:40:33.220] [info][status][plugin:markdown_vis@7.6.1] Status changed from uninitialized to green - Ready
:40:33.221] [info][status][plugin:metric_vis@7.6.1] Status changed from uninitialized to green - Ready
:40:33.222] [info][status][plugin:table_vis@7.6.1] Status changed from uninitialized to green - Ready
:40:33.226] [info][status][plugin:tagcloud@7.6.1] Status changed from uninitialized to green - Ready
:40:33.229] [info][status][plugin:vega@7.6.1] Status changed from uninitialized to green - Ready
:40:38.152] [warning][reporting] Generating a random key for xpack.reporting.encryptionKey. To prevent pen
s from failing on restart, please set xpack.reporting.encryptionKey in kibana.yml
:40:38.160] [info][status][plugin:reporting@7.6.1] Status changed from uninitialized to green - Ready
:40:38.189] [info][listening] Server running at http://localhost:5601
:40:38.390] [info][server][Kibana][http] http server running at http://localhost:5601

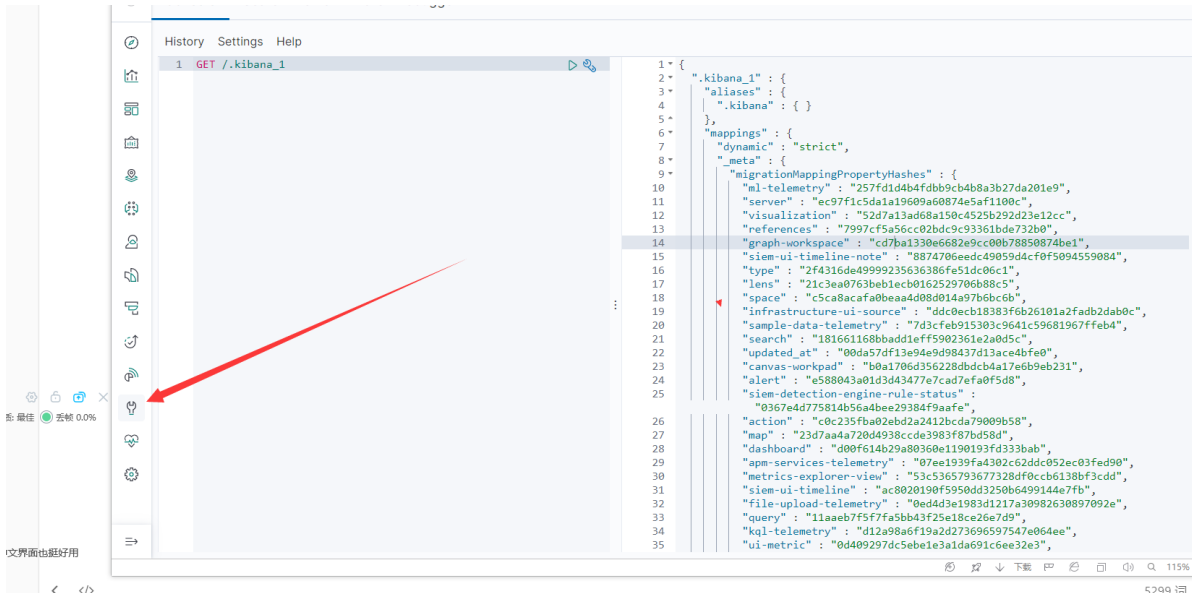
```

搜狗拼音输入法 全：

3、访问测试

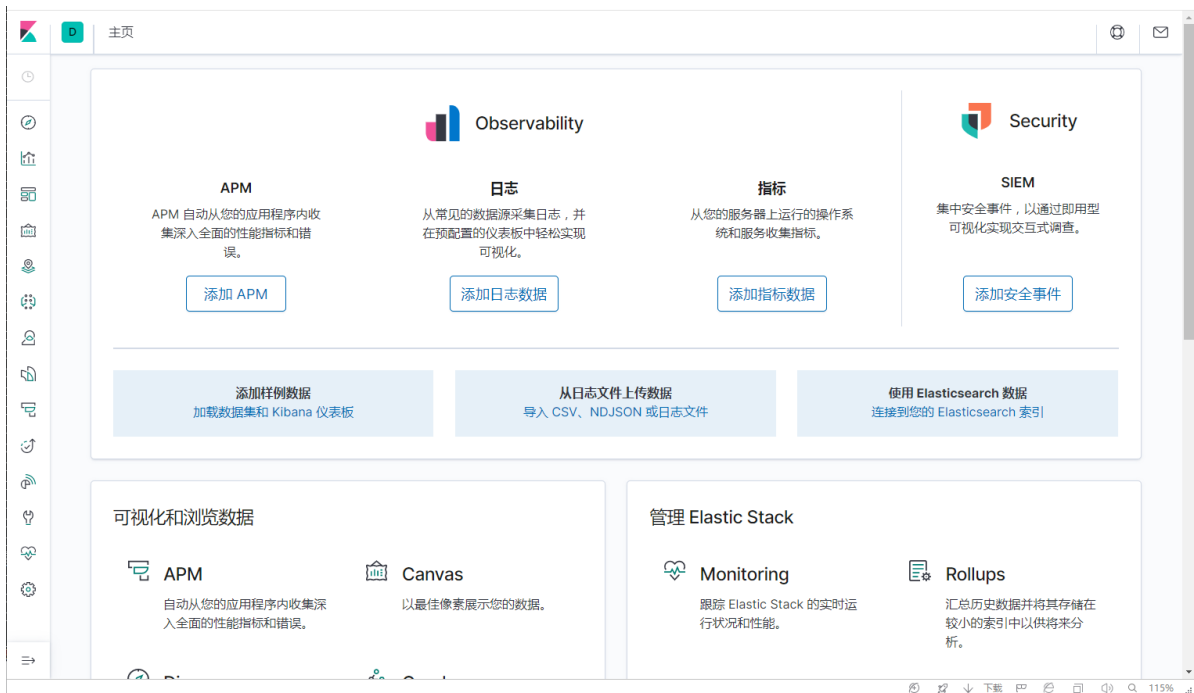


4、开发工具！（Post、curl、head、谷歌浏览器插件测试！）



我们之后的所有操作都在这里进行编写！很多学习大数据的人，Hadoop！

5、汉化！自己修改kibana配置即可！zh-CN！



ES核心概念

- 1、索引
- 2、字段类型 (mapping)
- 3、文档 (documents)

概述

在前面的学习中，我们已经掌握了es是什么，同时也把es的服务已经安装启动，那么es是如何去存储数据，数据结构是什么，又是如何实现搜索的呢？我们先来聊聊ElasticSearch的相关概念吧！

集群，节点，索引，类型，文档，分片，映射是什么？

elasticsearch是面向文档，关系行数据库 和 elasticsearch 客观的对比！一切都是JSON！

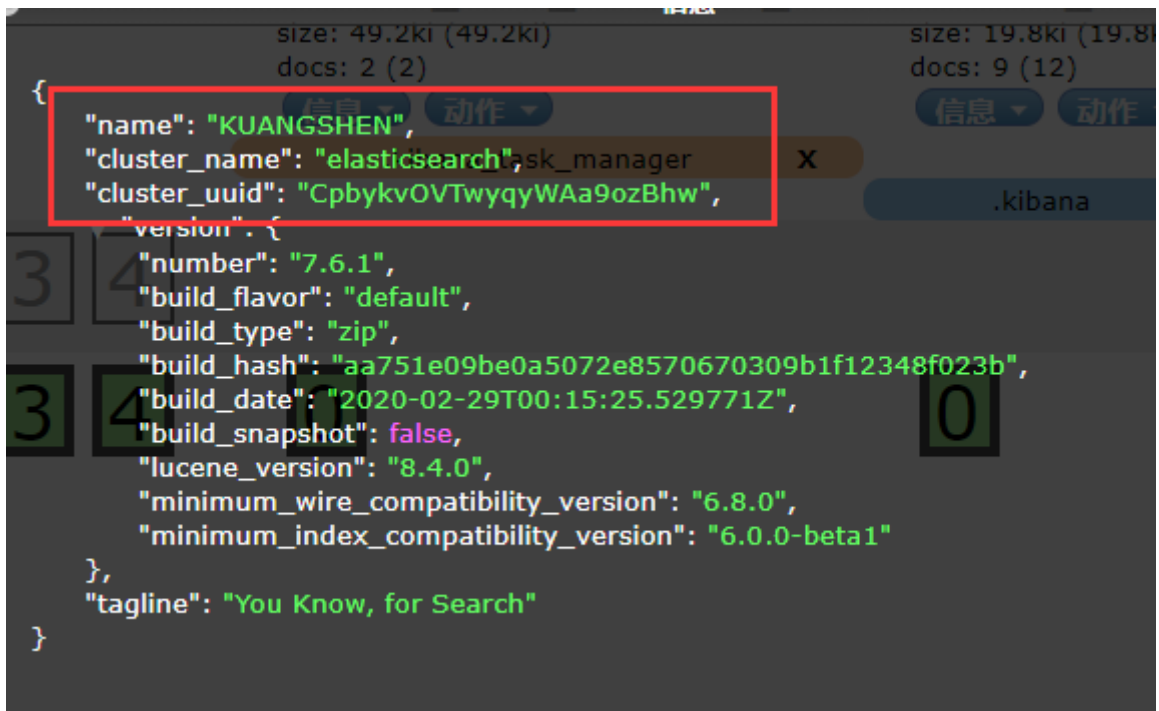
Relational DB	Elasticsearch
数据库(database)	索引(indices)
表(tables)	types
行(rows)	documents
字段(columns)	fields

elasticsearch(集群)中可以包含多个索引(数据库)，每个索引中可以包含多个类型(表)，每个类型下又包含多个文档(行)，每个文档中又包含多个字段(列)。

物理设计：

elasticsearch 在后台把每个**索引划分成多个分片**，每分片可以在集群中的不同服务器间迁移

一个人就是一个集群！默认的集群名称就是 elasticsearch



逻辑设计：

一个索引类型中，包含多个文档，比如说文档1，文档2。当我们索引一篇文档时，可以通过这样的一各顺序找到它：索引▷类型▷文档ID，通过这个组合我们就能索引到某个具体的文档。注意:ID不必是整数，实际上它是个字符串。

文档

就是我们的一条条数据

user		
1	zhangsan	18
2	kuangshen	3

之前说elasticsearch是面向文档的，那么就意味着索引和搜索数据的最小单位是文档，elasticsearch中，文档有几个重要属性：

- 自我包含，一篇文档同时包含字段和对应的值，也就是同时包含 key:value！
- 可以是层次型的，一个文档中包含自文档，复杂的逻辑实体就是这么来的！{就是一个json对象！fastjson进行自动转换！}
- 灵活的结构，文档不依赖预先定义的模式，我们知道关系型数据库中，要提前定义字段才能使用，在elasticsearch中，对于字段是非常灵活的，有时候，我们可以忽略该字段，或者动态的添加一个新的字段。

尽管我们可以随意的新增或者忽略某个字段，但是，每个字段的类型非常重要，比如一个年龄字段类型，可以是字符串也可以是整形。因为elasticsearch会保存字段和类型之间的映射及其他的设置。这种映射具体到每个映射的每种类型，这也是为什么在elasticsearch中，类型有时候也称为映射类型。

类型

核对 utf8_general_ci

列名	数据类型	长度	默认	主键?	非空?	Unsigned	自增?	Zerofill?	更新	注释
id	int	10		☒	☒	☐	☒	☐	☐	文章唯一id
author	varchar	50		☐	☒	☐	☐	☐	☐	作者
title	varchar	100		☐	☒	☐	☐	☐	☐	标题
content	longtext			☐	☒	☐	☐	☐	☐	文章的内容

类型是文档的逻辑容器，就像关系型数据库一样，表格是行的容器。类型中对于字段的定义称为映射，比如 name 映射为字符串类型。我们说文档是无模式的，它们不需要拥有映射中所定义的所有字段，比如新增一个字段，那么elasticsearch是怎么做的呢?elasticsearch会自动的将新字段加入映射，但是这个字段的不确定它是什么类型，elasticsearch就开始猜，如果这个值是18，那么elasticsearch会认为它是整形。但是elasticsearch也可能猜不对，所以最安全的方式就是提前定义好所需要的映射，这点跟关系型数据库殊途同归了，先定义好字段，然后再使用，别整什么幺蛾子。

索引

就是数据库！

索引是映射类型的容器，elasticsearch中的索引是一个非常大的文档集合。索引存储了映射类型的字段和其他设置。然后它们被存储到了各个分片上了。我们来研究下分片是如何工作的。

物理设计：节点和分片 如何工作

新建索引

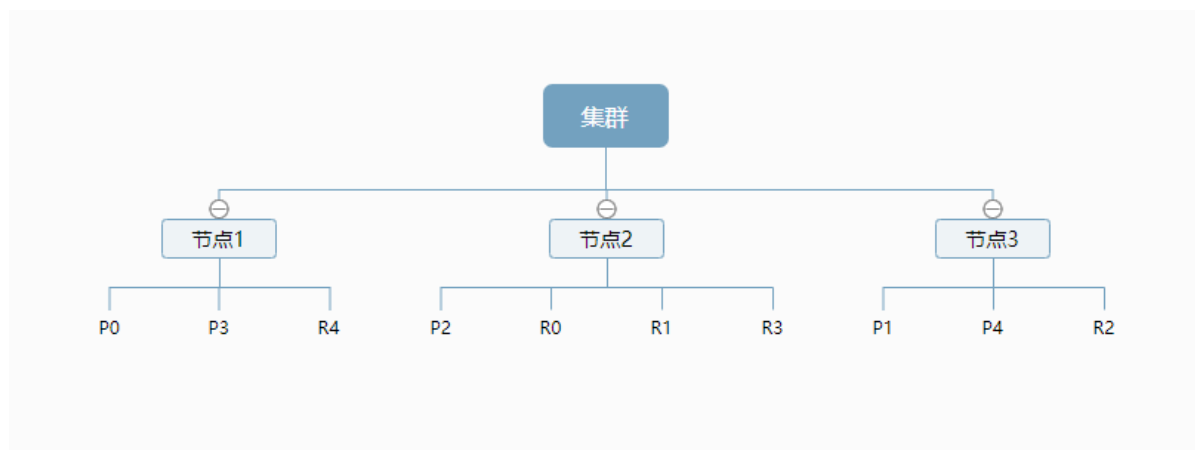
索引名称*

分片数* 5

副本数* 1

Cancel OK

一个集群至少有一个节点，而一个节点就是一个elasticsearch进程，节点可以有多个索引默认的，如果你创建索引，那么索引将会有个5个分片 (primary shard ,又称主分片) 构成的，每一个主分片会有一个副本 (replica shard ,又称复制分片)



上图是一个有3个节点的集群，可以看到主分片和对应的复制分片都不会在同一个节点内，这样有利于某个节点挂掉了，数据也不至于丢失。实际上，一个分片是一个Lucene索引，一个包含倒排索引的文件目录，倒排索引的结构使得elasticsearch在不扫描全部文档的情况下，就能告诉你哪些文档包含特定的关键字。不过，等等，倒排索引是什么鬼？

倒排索引

elasticsearch使用的是一种称为倒排索引的结构，采用Lucene倒排索引作为底层。这种结构适用于快速的全文搜索，一个索引由文档中所有不重复的列表构成，对于每一个词，都有一个包含它的文档列表。例如，现在有两个文档，每个文档包含如下内容：

Study every day, good good up to forever # 文档1包含的内容
To forever, study every day, good good up # 文档2包含的内容

为了创建倒排索引，我们首先要将每个文档拆分成独立的词(或称为词条或者tokens)，然后创建一个包含所有不重复的词条的排序列表，然后列出每个词条出现在哪个文档：

term	doc_1	doc_2
Study	√	×
To	×	×
every	√	√
forever	√	√
day	√	√
study	×	√
good	√	√
every	√	√
to	√	×
up	√	√

现在，我们试图搜索 to forever，只需要查看包含每个词条的文档 score

term	doc_1	doc_2
to	√	×
forever	√	√
total	2	1

两个文档都匹配，但是第一个文档比第二个匹配程度更高。如果没有别的条件，现在，这两个包含关键字的文档都将返回。

再来看一个示例，比如我们通过博客标签来搜索博客文章。那么倒排索引列表就是这样的一个结构：

博客文章(原始数据)		索引列表(倒排索引)	
博客文章ID	标签	标签	博客文章ID
1	python	python	1, 2, 3
2	python	linux	3, 4
3	linux, python		
4	linux		

如果要搜索含有 python 标签的文章，那相对于查找所有原始数据而言，查找倒排索引后的数据将会快的多。只需要 查看标签这一栏，然后获取相关的文章ID即可。完全过滤掉无关的所有数据，提高效率！

elasticsearch的索引和Lucene的索引对比

在elasticsearch中，索引（库）这个词被频繁使用，这就是术语的使用。在elasticsearch中，索引被分为多个分片，每份 分片是一个Lucene的索引。所以一个elasticsearch索引是由多个Lucene索引组成的。别问为什么，谁让elasticsearch使用Lucene作为底层呢！如无特指，说起索引都是指elasticsearch的索引。

接下来的一切操作都在kibana中Dev Tools下的Console里完成。基础操作！

IK分词器插件

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。

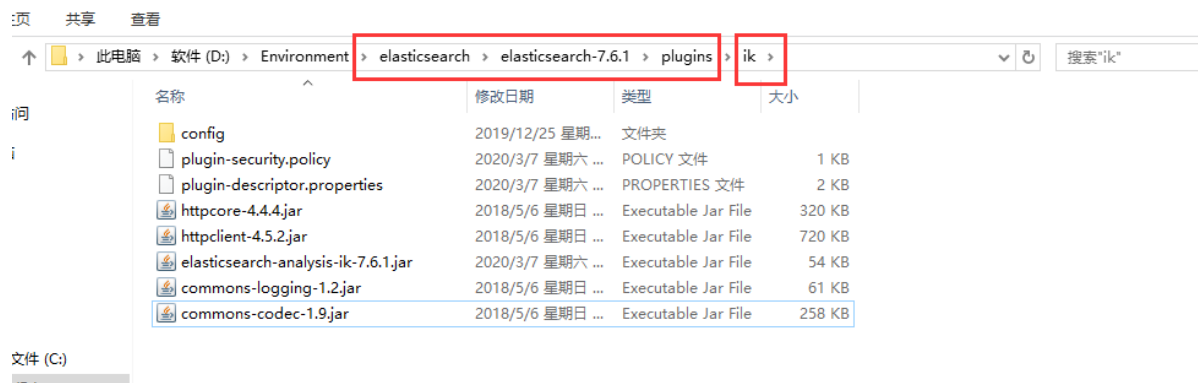
如果要使用中文，建议使用ik分词器！

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！

安装

1、<https://github.com/medcl/elasticsearch-analysis-ik>

2、下载完毕之后，放入到我们的elasticsearch 插件即可！



3、重启观察ES，可以看到ik分词器被加载了！

```
vice    ] [KUANGSHEN] loaded module [x-pack-voting-only-node]
vice    ] [KUANGSHEN] loaded module [x-pack-watcher]
vice    ] [KUANGSHEN] loaded plugin [analysis-ik]
```

4、elasticsearch-plugin 可以通过这个命令来查看加载进来的插件

```
D:\Environment\elasticsearch\elasticsearch-7.6.1\bin>elasticsearch-plugin list
future versions of Elasticsearch will require Java 11; your Java version from [D:\Environment\Java8\jdk8\jre] does not meet this requirement
ik
```

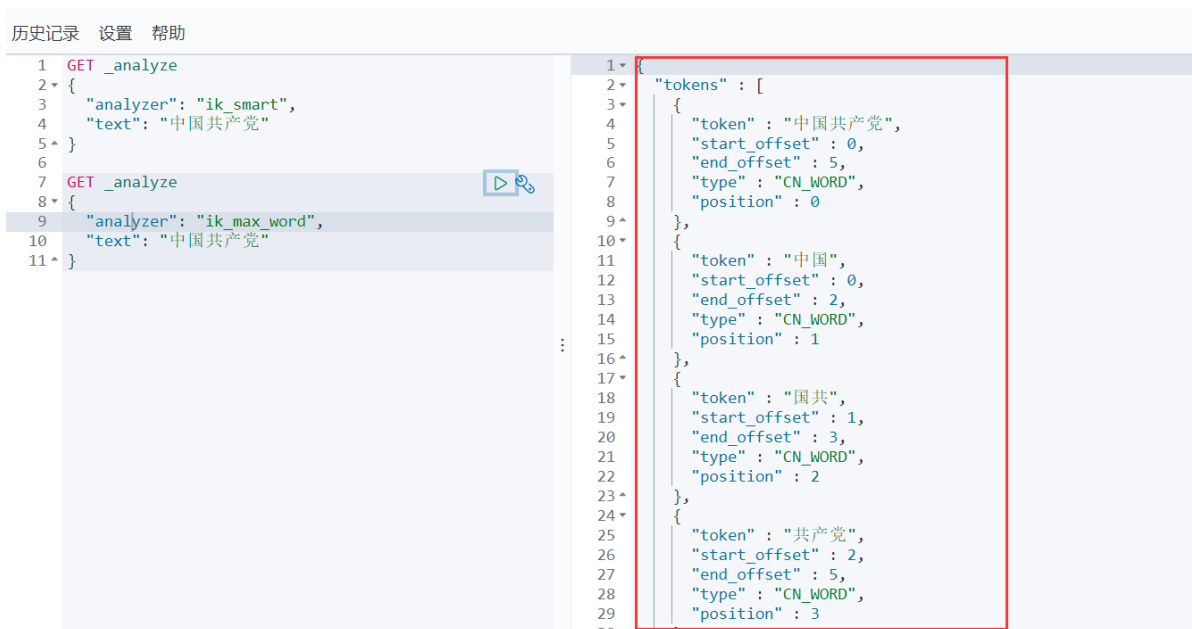
5、使用kibana测试！

查看不同的分词效果

其中 ik_smart 为最少切分



ik_max_word为最细粒度划分！穷尽词库的可能！字典！



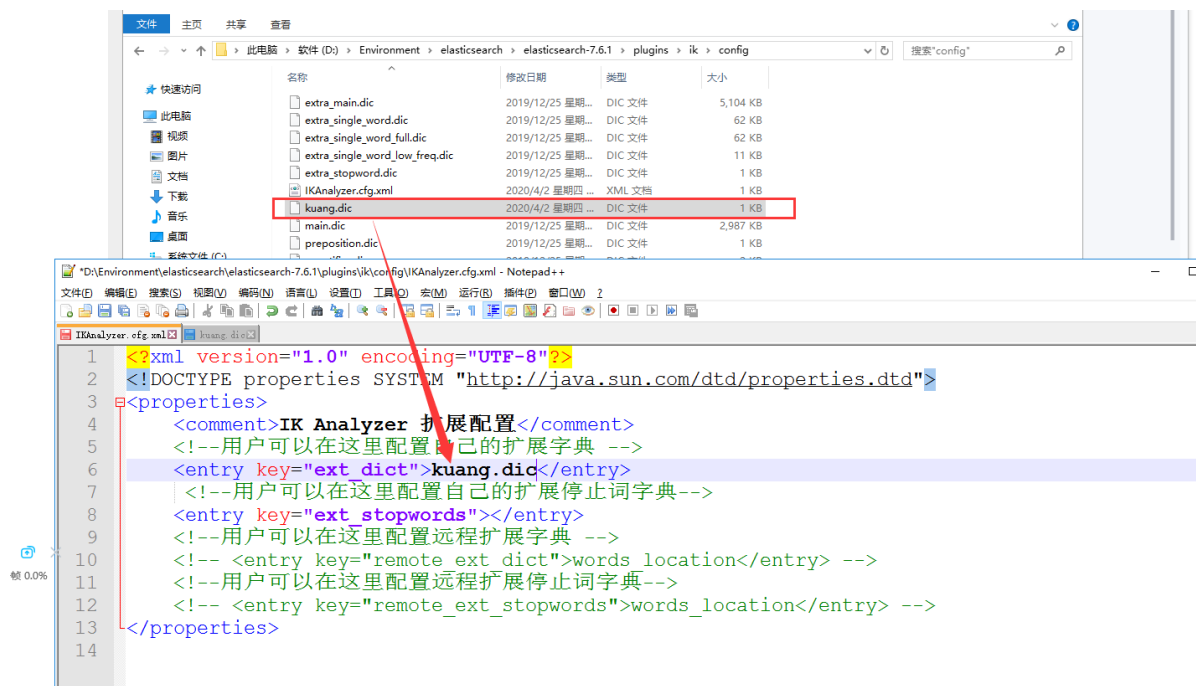
我们输入 超级喜欢狂神说Java



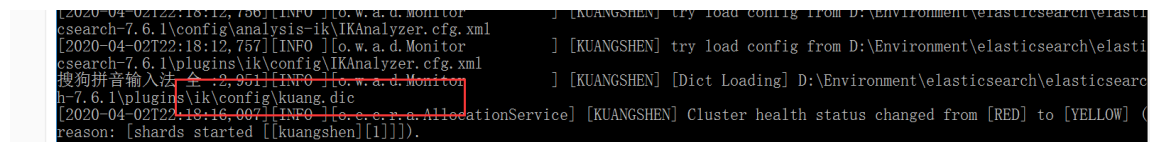
发现问题：狂神说被拆开了！

这种自己需要的词，需要自己加到我们的分词器的字典中！

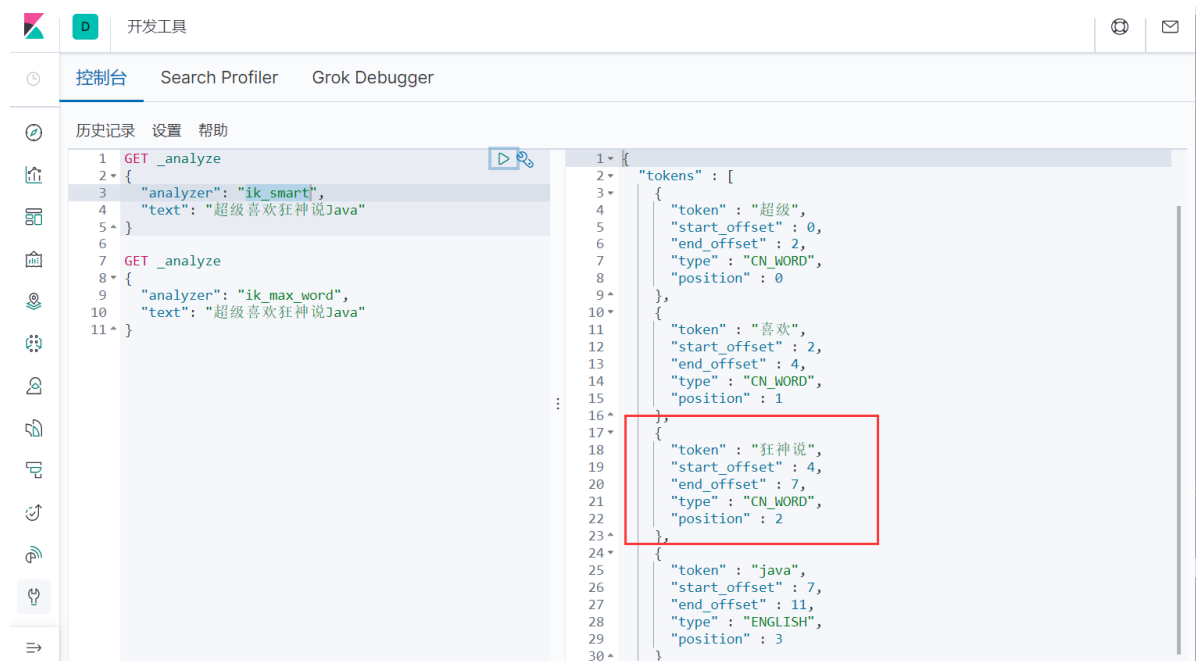
ik 分词器增加自己的配置！



重启es，看细节！



再次测试一下狂神说，看下效果！



以后的话，我们需要自己配置分词就在自己定义的dic文件中进行配置即可！

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端交互类的软件。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：

method	url地址	描述
PUT	localhost:9200/索引名称/类型名称/文档id	创建文档（指定文档id）
POST	localhost:9200/索引名称/类型名称	创建文档（随机文档id）
POST	localhost:9200/索引名称/类型名称/文档id/_update	修改文档
DELETE	localhost:9200/索引名称/类型名称/文档id	删除文档
GET	localhost:9200/索引名称/类型名称/文档id	查询文档通过文档id
POST	localhost:9200/索引名称/类型名称/_search	查询所有数据

关于索引的基本操作

1、创建一个索引！

PUT /索引名/~类型名~/文档id
{请求体}

历史记录 设置 帮助

```

1 PUT /test1/type1/1
2 {
3   "name": "狂神说",
4   "age": 3
5 }

```

```

1  #! Deprecation: [types removal] Specifying types in document index requests
   is deprecated, use the typeless endpoints instead (/index/_doc/{id},
   /index/_doc, or /index/_create/{id}).
2 {
3   "_index": "test1",
4   "_type": "type1",
5   "_id": "1",
6   "_version": 1,
7   "result": "created",
8   "shards": {
9     "total": 2,
10    "successful": 1,
11    "failed": 0
12  },
13   "seq_no": 0,
14   "_primary_term": 1
15 }
16

```

完成了自动增加了索引！数据也成功的添加了，这就是我说大家在初期可以把它当做数据库学习的原因！

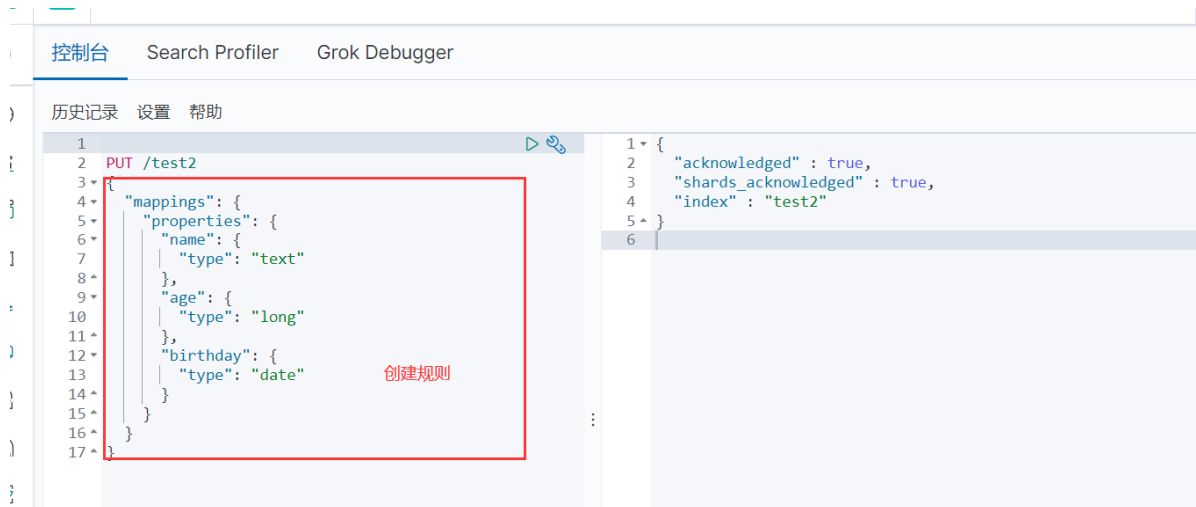


3、那么 name 这个字段用不用指定类型呢。毕竟我们关系型数据库 是需要指定类型的啊！

- 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.....

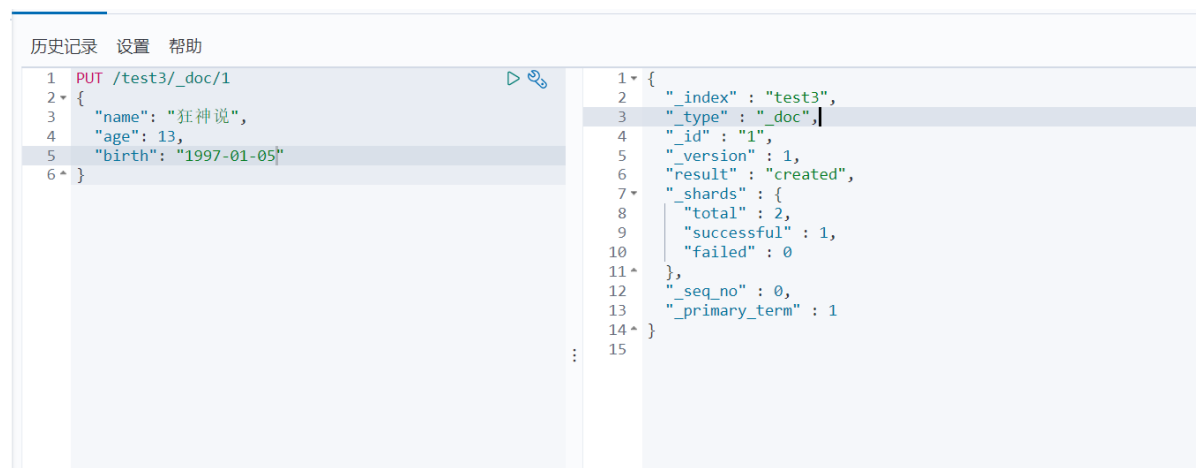
4、指定字段的类型

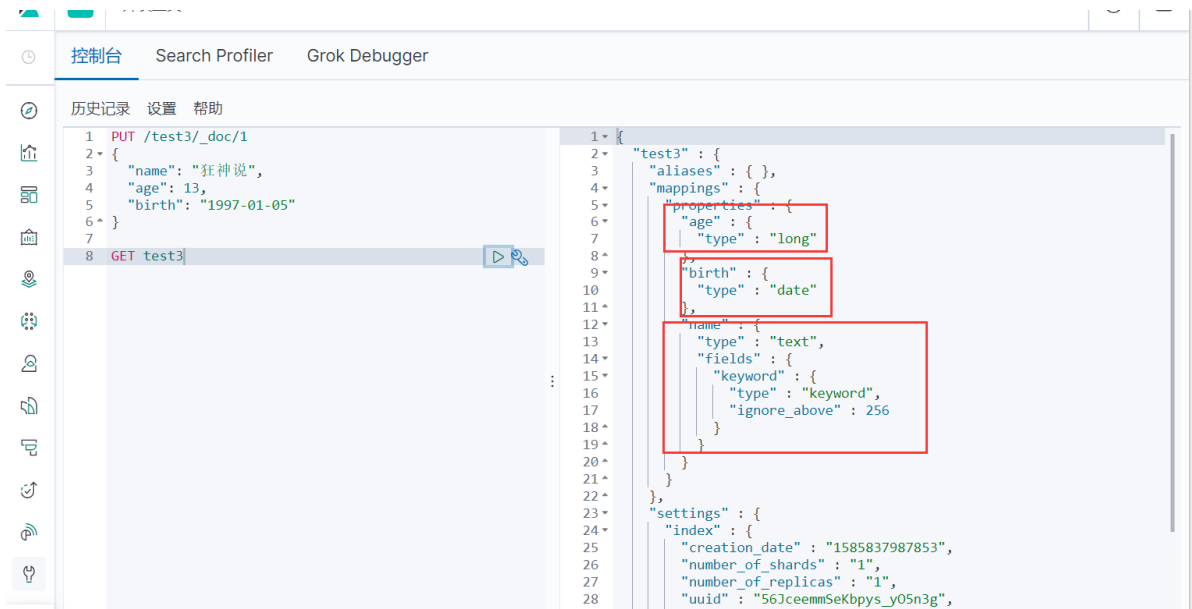


获得这个规则！可以通过 GET 请求获取具体的信息！



5、查看默认的信息

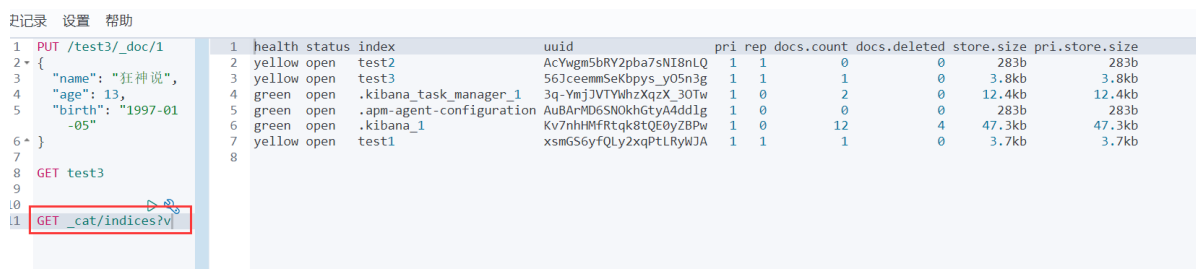




如果自己的文档字段没有指定，那么es 就会给我们默认配置字段类型！

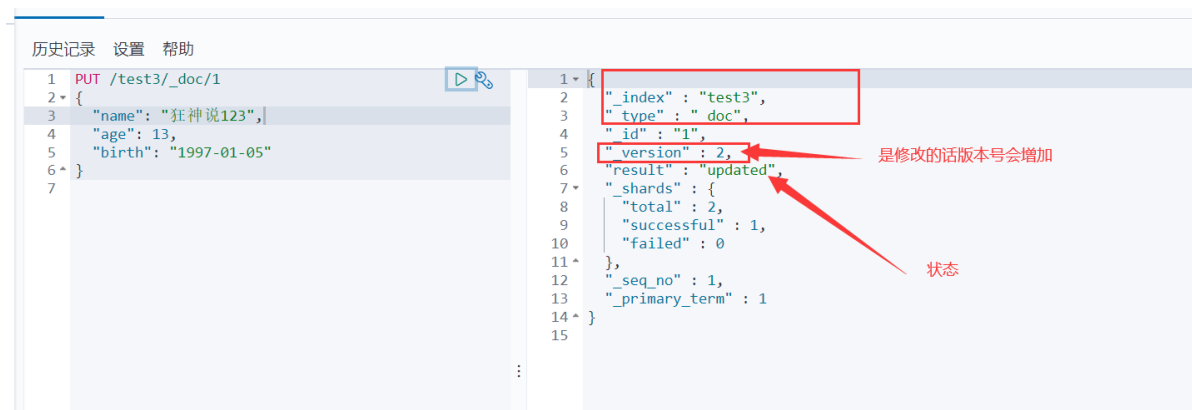
虚心学习，这个世界上大佬很多！

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！

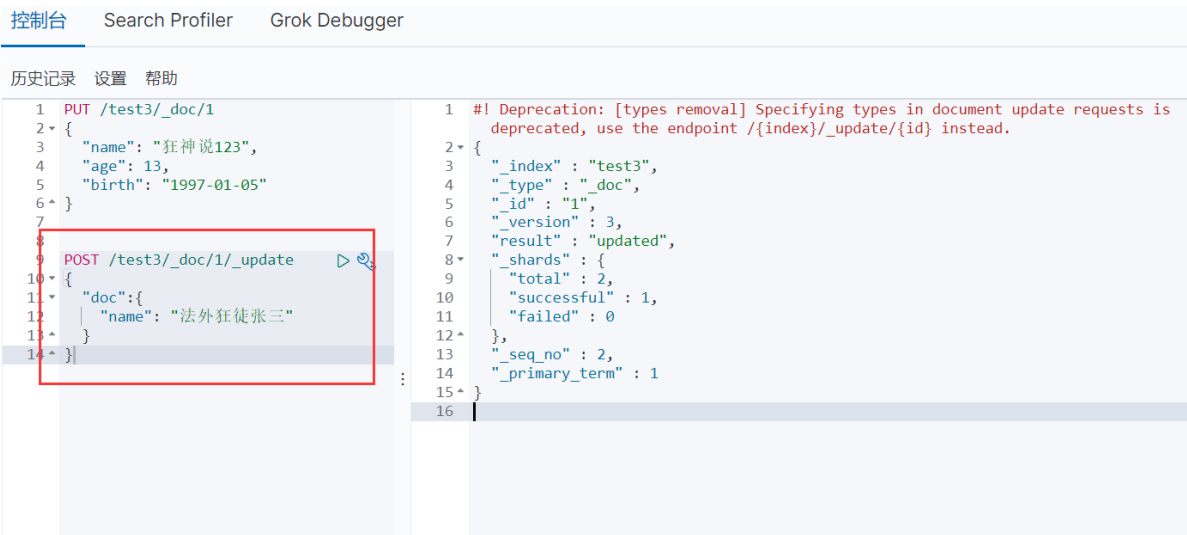


修改 提交还是使用PUT 即可！然后覆盖！最新办法！

曾经！



现在的方法！



删除索引！

通过DELETE 命令实现删除、 根据你的请求来判断是删除索引还是删除文档记录！

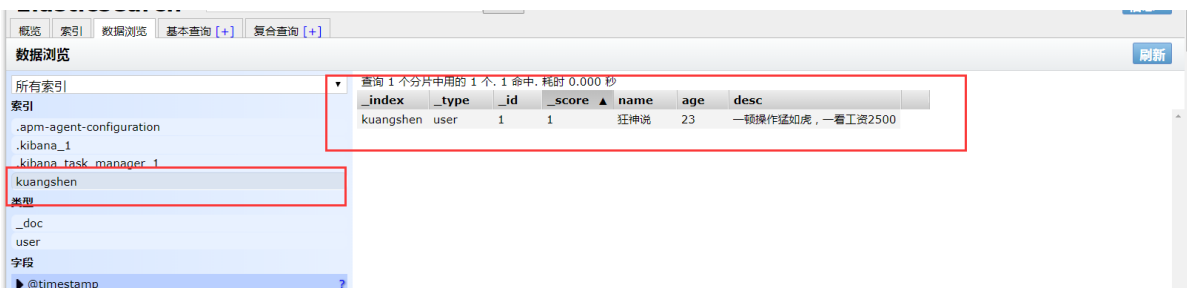
使用RESTFUL 风格是我们ES推荐大家使用的！

关于文档的基本操作（重点）

基本操作

1、添加数据

```
PUT /kuangshen/user/1
{
  "name": "狂神说",
  "age": 23,
  "desc": "一顿操作猛如虎，一看工资2500",
  "tags": ["技术宅", "温暖", "直男"]
}
```



2、获取数据 GET

历史记录 设置 帮助

```

1 PUT /kuangshen/user/3
2 {
3   "name": "李四",
4   "age": 30,
5   "desc": "mmp, 不知道如何形容",
6   "tags": ["靓女", "旅游", "唱歌"]
7 }
8
9
10 GET kuangshen/user/1

```

```

1 #! Deprecation: [types removal] Specifying types in document get requests is
  deprecated, use the /{index}/_doc/{id} endpoint instead.
2 {
3   "_index": "kuangshen",
4   "_type": "user",
5   "_id": "1",
6   "_version": 1,
7   "_seq_no": 0,
8   "_primary_term": 1,
9   "found": true,
10  "source": {
11    "name": "狂神说",
12    "age": 23,
13    "desc": "一顿操作猛如虎，一看工资2500",
14    "tags": [
15      "技术宅",
16      "温暖",
17      "直男"
18    ]
19  }
20 }
21

```

3、更新数据 PUT

```

1 PUT /kuangshen/user/3
2 {
3   "name": "李四233",
4   "age": 30,
5   "desc": "mmp, 不知道如何形容",
6   "tags": ["靓女", "旅游", "唱歌"]
7 }
8
9
10 GET kuangshen/user/1

```

```

1 #! Deprecation: [types removal] Specifying types in document index requests is
  deprecated, use the typeless endpoints instead (/_{index}/_doc/{id}, /_{index}
  /_doc, or /_{index}/_create/{id}).
2 {
3   "_index": "kuangshen",
4   "_type": "user",
5   "_id": "3",
6   "version": 2,
7   "result": "updated",
8   "shards": {
9     "total": 2,
10    "successful": 1,
11    "failed": 0
12  },
13   "_seq_no": 3,
14   "_primary_term": 1
15 }
16

```

这里的version就代表这个数据被改动的次数

4、Post _update, 推荐使用这种更新方式！

```

1 PUT /kuangshen/user/1
2 {
3   "name": "狂神说",
4   "age": 23,
5   "desc": "一顿操作猛如虎，一看工资2500",
6   "tags": ["技术宅", "温暖", "直男"]
7 }
8
9
10 GET kuangshen/user/1

```

put如果不传递值
就会被覆盖

```

13 POST kuangshen/user/1/_update
14 {
15   "doc": {
16     "name": "狂神说Java"
17   }
18 }

```

灵活性更好！

```

1 #! Deprecation: [types removal] Specifying types in document update requests is
  deprecated, use the endpoint /_{index}/_update/{id} instead.
2 {
3   "_index": "kuangshen",
4   "_type": "user",
5   "_id": "1",
6   "_version": 4,
7   "result": "updated",
8   "shards": {
9     "total": 2,
10    "successful": 1,
11    "failed": 0
12  },
13   "_seq_no": 6,
14   "_primary_term": 1
15 }
16

```

简单地搜索！

GET kuangshen/user/1

简答的条件查询，可以根据默认的映射规则，产生基本的查询！

开发工具

控制台 Search Profiler Grok Debugger

历史记录 设置 帮助

```
1 PUT /kuangshen/user/1
2 {
3   "name": "狂神说Java",
4   "age": 23,
5   "desc": "一顿操作猛如虎，一看工资2500",
6   "tags": ["技术宅", "温暖", "直男"]
7 }
8
9
10 GET kuangshen/user/_search?q=name:狂神说
11
12
13 GET kuangshen/user/_search?q=name:狂神说Java
```

查询条件

```
1 #! Deprecation: [types removal] Specifying types in search requests is deprecated.
2 {
3   "took": 0,
4   "timed_out": false,
5   "_shards": {
6     "total": 1,
7     "successful": 1,
8     "skipped": 0,
9     "failed": 0
10  },
11  "hits": {
12    "total": {
13      "value": 1,
14      "relation": "eq"
15    },
16    "max_score": 1.300138,
17    "hits": [
18      {
19        "_index": "kuangshen",
20        "_type": "user",
21        "_id": "1",
22        "_score": 1.300138,
23        "_source": {
24          "name": "狂神说Java",
25          "age": 23,
26          "desc": "一顿操作猛如虎，一看工资2500",
27          "tags": ["技术宅", "温暖", "直男"]
28        }
29      }
30    ]
31  }
32 }
```

控制台 Search Profiler Grok Debugger

历史记录 设置 帮助

```
1 PUT /kuangshen/user/1
2 {
3   "name": "狂神说Java",
4   "age": 23,
5   "desc": "一顿操作猛如虎，一看工资2500",
6   "tags": ["技术宅", "温暖", "直男"]
7 }
8
9
10 GET kuangshen/user/_search?q=name:狂神说
11
12
13 GET kuangshen/user/_search?q=name:狂神说Java
```

```
7   "successful": 1,
8   "skipped": 0,
9   "failed": 0
10  },
11  "hits": {
12    "total": {
13      "value": 1,
14      "relation": "eq"
15    },
16    "max_score": 2.4273598,
17    "hits": [
18      {
19        "_index": "kuangshen",
20        "_type": "user",
21        "_id": "1",
22        "_score": 2.4273598,
23        "_source": {
24          "name": "狂神说Java",
25          "age": 23,
26          "desc": "一顿操作猛如虎，一看工资2500",
27          "tags": [
28            "技术宅",
29            "温暖",
30            "直男"
31          ]
32        }
33      }
34    ]
35  }
```

未来如果存在多条查询出来的结果！
匹配度，匹配度越高则分值越高

复杂操作搜索 select (排序, 分页, 高亮, 模糊查询, 精准查询!)

历史记录 设置 帮助

```
1 GET kuangshen/user/_search
2 {
3   "query": {
4     "match": {
5       "name": "狂神"
6     }
7   }
8 }
```

查询的参数体使用Json构

```
历史记录 设置 帮助
1 GET kuangshen/user/_search
2 {
3   "query": {
4     "match": {
5       "name": "狂神"
6     }
7   }
8 }
9

14 {
15   "relation": "eq"
16 },
17   "max_score": 1.3097506,
18   "hits": [
19     {
20       "_index": "kuangshen",
21       "_type": "user",
22       "_id": "1",
23       "_score": 1.3097506,
24       "_source": {
25         "name": "狂神说Java",
26         "age": 23,
27         "desc": "一顿操作猛如虎，一看工资2500",
28         "tags": [
29           "技术宅",
30           "温暖",
31           "直男"
32         ]
33       }
34     },
35     {
36       "_index": "kuangshen",
37       "_type": "user",
38       "_id": "4",
39       "_score": 1.179499,
40       "_source": {
41         "name": "狂神说前端",
42         "age": 3,
43         "desc": "一顿操作猛如虎，一看工资2500",
44         "tags": [
45           "技术宅",
46           "温暖",
47           "直男"
48         ]
49       }
50     }
51   ]
52 }
```

hit:
索引和文档的信息
查询的结果总数
然后就是查询出来的具体的文档
数据中的东西都可以遍历出来了
分数：我们可以通过判断谁更加符合结果

输出结果，不想要那么多！

```
历史记录 设置 帮助
1 GET kuangshen/user/_search
2 {
3   "query": {
4     "match": {
5       "name": "狂神"
6     }
7   }
8 }
9   "_source": ["name", "desc"]
10 }
```

结果的过滤

```
13 {
14   "value": 2,
15   "relation": "eq"
16 },
17   "max_score": 1.3097506,
18   "hits": [
19     {
20       "_index": "kuangshen",
21       "_type": "user",
22       "_id": "1",
23       "_score": 1.3097506,
24       "_source": {
25         "name": "狂神说Java",
26         "desc": "一顿操作猛如虎，一看工资2500"
27       }
28     },
29     {
30       "_index": "kuangshen",
31       "_type": "user",
32       "_id": "4",
33       "_score": 1.179499,
34       "_source": {
35         "name": "狂神说前端",
36         "desc": "一顿操作猛如虎，一看工资2500"
37       }
38     }
39   ]
40 }
```

我们之后使用Java操作es，所有的方法和对象就是这里面的 key！

排序！

```
历史记录 设置 帮助
1 GET kuangshen/user/_search
2 {
3   "query": {
4     "match": {
5       "name": "狂神"
6     }
7   }
8 }
9   "sort": [
10     {
11       "age": {
12         "order": "asc"
13       }
14     }
15   ]
16 }
```

通过那个字段进行排序

```
21   "_id": "4",
22   "_score": null,
23   "_source": {
24     "name": "狂神说前端",
25     "age": 3,
26     "desc": "一顿操作猛如虎，一看工资2500",
27     "tags": [
28       "技术宅",
29       "温暖",
30       "直男"
31     ]
32   },
33   "sort": [
34     3
35   ]
36 },
37 {
38   "_index": "kuangshen",
39   "_type": "user",
40   "_id": "1",
41   "_score": null,
42   "_source": {
43     "name": "狂神说Java",
44     "age": 23,
45     "desc": "一顿操作猛如虎，一看工资2500",
46     "tags": [
47       "技术宅",
48       "温暖",
49       "直男"
50     ]
51   }
52 }
```

分页查询！

```
历史记录 设置 帮助
1 GET kuangshen/user/_search
2 {
3   "query": {
4     "match": {
5       "name": "狂神"
6     }
7   },
8   "sort": [
9     {
10      "age": {
11        "order": "asc"
12      }
13    }
14  ],
15  "from": 0,
16  "size": 1
17 }
18 }

1 #! Deprecation: [types removal] Specifying types in search requests is deprecated.
2 {
3   "took": 0,
4   "timed_out": false,
5   "_shards": {
6     "total": 1,
7     "successful": 1,
8     "skipped": 0,
9     "failed": 0
10  },
11  "hits": {
12    "total": {
13      "value": 2,
14      "relation": "eq"
15    },
16    "max_score": null,
17    "hits": [
18      {
19        "_index": "kuangshen",
20        "_type": "user",
21        "_id": "4",
22        "_score": null,
23        "_source": {
24          "name": "狂神说前端",
25          "age": 3,
```

数据下标还是从0开始的，和学的所有数据结构是一样的！

/search/{current}/{pagesize}

布尔值查询

must (and) , 所有的条件都要符合 where id = 1 and name = xxx

```
历史记录 设置 帮助
1 GET kuangshen/user/_search
2 {
3   "query": {
4     "bool": {
5       "must": [
6         {
7           "match": {
8             "name": "狂神说"
9           }
10        },
11        {
12           "match": {
13             "age": 23
14           }
15        }
16      ]
17    }
18  }
19 }
20 }

9 "failed": 0
10 },
11 "hits": {
12   "total": {
13     "value": 1,
14     "relation": "eq"
15   },
16   "max_score": 2.9646258,
17   "hits": [
18     {
19       "_index": "kuangshen",
20       "_type": "user",
21       "_id": "1",
22       "_score": 2.9646258,
23       "_source": {
24         "name": "狂神说Java",
25         "age": 23,
26         "desc": "一顿操作猛如虎，一看工资2500",
27         "tags": [
28           "技术宅",
29           "温暖",
30           "直男"
31         ]
32       }
33     }
34   ]
35 }
```

should (or) , 所有的条件都要符合 where id = 1 or name = xxx

```
1 GET kuangshen/user/_search
2 {
3   "query": {
4     "bool": {
5       "should": [
6         {
7           "match": {
8             "name": "狂神说"
9           }
10        },
11        {
12           "match": {
13             "age": 23
14           }
15        }
16      ]
17    }
18  }
19 }
20 }

14 "relation": "eq"
15 },
16 "max_score": 2.9646258,
17 "hits": [
18   {
19     "_index": "kuangshen",
20     "_type": "user",
21     "_id": "1",
22     "_score": 2.9646258,
23     "_source": {
24       "name": "狂神说Java",
25       "age": 23,
26       "desc": "一顿操作猛如虎，一看工资2500",
27       "tags": [
28         "技术宅",
29         "温暖",
30         "直男"
31       ]
32     }
33   },
34   {
35     "_index": "kuangshen",
36     "_type": "user",
37     "_id": "4",
38     "_score": 1.7692485,
```

must_not (not)

```
历史记录 设置 帮助
1 GET kuangshen/user/_search
2 {
3   "query": {
4     "bool": {
5       "must_not": [
6         {
7           "match": {
8             "age": 3
9           }
10        }
11      ]
12    }
13  }
14 }
15 }

11 "hits": {
12   "total": {
13     "value": 2,
14     "relation": "eq"
15   },
16   "max_score": 0.0,
17   "hits": [
18     {
19       "_index": "kuangshen",
20       "_type": "user",
21       "_id": "3",
22       "_score": 0.0,
23       "_source": {
24         "name": "李四233",
25         "age": 30,
26         "desc": "mmp, 不知道如何形容",
27         "tags": [
28           "靓女",
29           "旅游",
30           "眼瞎"
```

过滤器 filter

```
历史记录 设置 帮助
1 GET kuangshen/user/_search
2 {
3   "query": {
4     "bool": {
5       "must": [
6         {
7           "match": {
8             "name": "狂神"
9           }
10        }
11      ],
12      "filter": {
13        "range": {
14          "age": {
15            "lt": 10
16          }
17        }
18      }
19    }
20  }
21 }
22 }

9 "failed": 0
10 },
11 "hits": {
12   "total": {
13     "value": 1,
14     "relation": "eq"
15   },
16   "max_score": 1.179499,
17   "hits": [
18     {
19       "_index": "kuangshen",
20       "_type": "user",
21       "_id": "4",
22       "_score": 1.179499,
23       "_source": {
24         "name": "狂神说前端",
25         "age": 3,
26         "desc": "一顿操作猛如虎，一看工资2500",
27         "tags": [
28           "技术宅",
29           "温暖",
30           "直男"
31         ]
32       }
33     }
34   ]
35 }
36 }
```

- gt 大于
- gte 大于等于
- lt 小于
- lte 小于等于！

```
历史记录 设置 帮助
1 GET kuangshen/user/_search
2 {
3   "query": {
4     "bool": {
5       "must": [
6         {
7           "match": {
8             "name": "狂神"
9           }
10        }
11      ],
12      "filter": {
13        "range": {
14          "age": {
15            "gte": 1,
16            "lte": 25
17          }
18        }
19      }
20    }
21  }
22 }
23 }

18 {
19   "_index": "kuangshen",
20   "_type": "user",
21   "_id": "1",
22   "_score": 1.3097506,
23   "_source": {
24     "name": "狂神说Java",
25     "age": 23,
26     "desc": "一顿操作猛如虎，一看工资2500",
27     "tags": [
28       "技术宅",
29       "温暖",
30       "直男"
31     ]
32   }
33 },
34 {
35   "_index": "kuangshen",
36   "_type": "user",
37   "_id": "4",
38   "_score": 1.179499,
39   "_source": {
40     "name": "狂神说前端",
41     "age": 3,
```

匹配多个条件！

```
历史记录 设置 帮助
1 GET kuangshen/user/_search
2 {
3   "query": {
4     "match": {
5       "tags": "男 技术"
6     }
7   }
8 }
9

多个条件使用空格隔开
只要满足其中一个结果就可以被查出
这个时候可以通过分值基本的判断

34 {
35   "_index": "kuangshen",
36   "_type": "user",
37   "_id": "4",
38   "_score": 1.689794,
39   "source": {
40     "name": "狂神说前端",
41     "age": 3,
42     "desc": "一顿操作猛如虎，一看工资2500",
43     "tags": [
44       "技术宅",
45       "温暖",
46       "直男"
47     ]
48   }
49 },
50 {
51   "_index": "kuangshen",
52   "_type": "user",
53   "_id": "2",
54   "_score": 0.36826363,
55   "source": {
56     "name": "张三",
57     "age": 3,
58     "desc": "法外狂徒",
59     "tags": [
60       "交友",
61       "旅游",
62       "渣男"
```

精确查询！

term 查询是直接通过倒排索引指定的词条进行精确查找的！

关于分词：

- term，直接查询精确的
- 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 keyword

```
历史记录 设置 帮助
22 PUT testdb/_doc/2
23 {
24   "name": "狂神说Java name",
25   "desc": "狂神说Java desc2"
26 }
27
28 GET _analyze
29 {
30   "analyzer": "keyword",
31   "text": "狂神说Java name"
32 }
33
34 GET _analyze
35 {
36   "analyzer": "standard",
37   "text": "狂神说Java name"
38 }
39
40
41
42
43
44
45

1 {
2   "tokens": [
3     {
4       "token": "狂神说Java name",
5       "start_offset": 0,
6       "end_offset": 12,
7       "type": "word",
8       "position": 0
9     }
10  ]
11 }
12
```

```
历史记录 设置 帮助
29 {
30   "analyzer": "keyword",
31   "text": "狂神说Java name"
32 }
33
34 GET _analyze
35 {
36   "analyzer": "standard",
37   "text": "狂神说Java name"
38 }
39
40
41
42
43
44
45
46
47
48
49

1 {
2   "tokens": [
3     {
4       "token": "狂",
5       "start_offset": 0,
6       "end_offset": 1,
7       "type": "<IDEOGRAPHIC>",
8       "position": 0
9     },
10    {
11      "token": "神",
12      "start_offset": 1,
13      "end_offset": 2,
14      "type": "<IDEOGRAPHIC>",
15      "position": 1
16    },
17    {
18      "token": "说",
19      "start_offset": 2,
20      "end_offset": 3,
```

```
历史记录 设置 帮助
19 "desc": "狂神说Java desc"
20 }
21
22 PUT testdb/_doc/2
23 {
24   "name": "狂神说Java name",
25   "desc": "狂神说Java desc2"
26 }
27
28 GET testdb/_search
29 {
30   "query": {
31     "term": {
32       "name": "狂"
33     }
34   }
35 }
36
37 GET testdb/_search
38 {
39   "query": {
40     "term": {
41       "desc": "狂神说Java desc"
42     }
43   }
44 }
45
46
2 "took": 0,
3 "timed_out": false,
4 "_shards": {
5   "total": 1,
6   "successful": 1,
7   "skipped": 0,
8   "failed": 0
9 },
10 "hits": {
11   "total": {
12     "value": 1,
13     "relation": "eq"
14   },
15   "max_score": 0.6931471,
16   "hits": [
17     {
18       "_index": "testdb",
19       "_type": "_doc",
20       "_id": "1",
21       "_score": 0.6931471,
22       "_source": {
23         "name": "狂神说Java name",
24         "desc": "狂神说Java desc"
25       }
26     }
27   ]
28 }
29
```

多个值匹配精确查询

```
历史记录 设置 帮助
3 "t1": "22",
4 "t2": "2020-4-6"
5 }
6 PUT testdb/_doc/4
7 {
8   "t1": "33",
9   "t2": "2020-4-7"
10 }
11
12
13 GET testdb/_search
14 {
15   "query": {
16     "bool": {
17       "should": [
18         {
19           "term": {
20             "t1": "22"
21           }
22         },
23         {
24           "term": {
25             "t1": "33"
26           }
27         }
28       ]
29     }
30   }
31 }
32
33
34
35
36
37
38
39
12 "value": 2,
13 "relation": "eq"
14 },
15 "max_score": 0.6931471,
16 "hits": [
17   {
18     "_index": "testdb",
19     "_type": "_doc",
20     "_id": "3",
21     "_score": 0.6931471,
22     "_source": {
23       "t1": "22",
24       "t2": "2020-4-6"
25     }
26   },
27   {
28     "_index": "testdb",
29     "_type": "_doc",
30     "_id": "4",
31     "_score": 0.6931471,
32     "_source": {
33       "t1": "33",
34       "t2": "2020-4-7"
35     }
36   }
37 ]
38 }
39
```

高亮查询！

```
历史记录 设置 帮助
1 GET kuangshen/user/_search
2 {
3   "query": {
4     "match": {
5       "name": "狂神说"
6     }
7   },
8   "highlight": {
9     "fields": {
10       "name": {}
11     }
12   }
13 }
14
15
16
17
18
19
20
21
22
23
24
25 "age": 23,
26 "desc": "一顿操作猛如虎，一看工资2500",
27 "tags": [
28   "技术宅",
29   "温暖",
30   "直男"
31 ],
32 "highlight": {
33   "name": [
34     "<em>狂</em><em>神</em>说</em>Java"
35   ]
36 }
37
38
39
40 "_index": "kuangshen",
41 "_type": "user",
42 "_id": "4",
43 "_score": 1.7692485,
44 "_source": {
45   "name": "狂神说前端",
46   "age": 23,
47   "desc": "一顿操作猛如虎，一看工资2500",
48   "tags": [
49     "技术宅",
50     "温暖",
51     "直男"
52 ]
53 }
54 }
```



```
1 GET kuangshen/user/_search
2 {
3   "query": {
4     "match": {
5       "name": "狂神说"
6     }
7   },
8   "highlight": {
9     "pre_tags": "<p class='key' style='color:red'>",
10    "post_tags": "</p>",
11    "fields": {
12      "name": {}
13    }
14  }
15 }
```

```
36 {
37   }
38 },
39 {
40   "_index": "kuangshen",
41   "_type": "user",
42   "_id": "4",
43   "_score": 1.7692485,
44   "_source": {
45     "name": "狂神说前端",
46     "age": 3,
47     "desc": "一顿操作猛如虎，一看工资2500",
48     "tags": [
49       "技术宅",
50       "温暖",
51       "直男"
52     ]
53   },
54   "highlight": {
55     "name": [
56       "<p class='key' style='color:red'>狂</p><p class='key' style='color:red'>神</p><p class='key' style='color:red'>说</p>前端"
57     ]
58   }
59 }
60 }
61 }
62 }
```

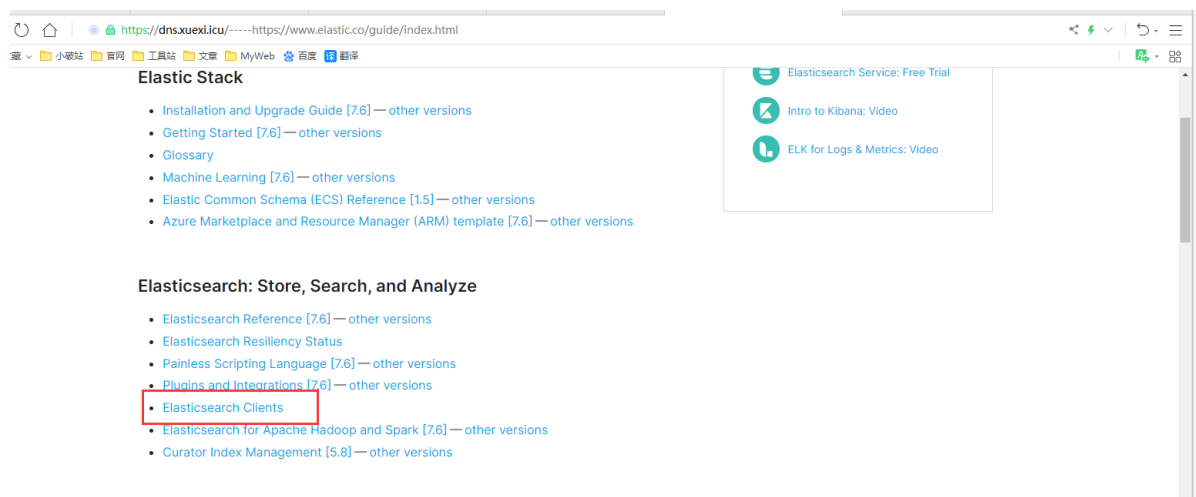
自定义搜索高亮条件

这些其实MySQL 也可以做，只是MySQL 效率比较低！

- 匹配
- 按照条件匹配
- 精确匹配
- 区间范围匹配
- 匹配字段过滤
- 多条件查询
- 高亮查询

集成SpringBoot

找官方文档！



elastic Products Learn Company Pricing Contact Try Free Login

Docs

Elasticsearch Clients

- Java REST Client [7.6] — other versions 推荐使用
- Java API [7.6] — other versions
- JavaScript API [7.x] — other versions
- Ruby API [7.x] — other versions
- Go API
- PHP API [7.x] — other versions
- Python API
- eland Client
- Rust API
- Community Contributed Clients

Most Popular

- Elasticsearch Service: Free Trial
- Intro to Kibana: Video
- ELK for Logs & Metrics: Video

elastic Products Learn Company Pricing Contact Try Free Login

Docs

Overview »

Java REST Client

+ Java REST Client: 7.6 (current)

Overview

- Java Low Level REST Client
- Java High Level REST Client 我们一般使用高级的客户端！

Overview »

Most Popular

- Elasticsearch Service: Free Trial
- Intro to Kibana: Video
- ELK for Logs & Metrics: Video

1、找到原生的依赖

```
<dependency>
  <groupId>org.elasticsearch.client</groupId>
  <artifactId>elasticsearch-rest-high-level-client</artifactId>
  <version>7.6.2</version>
</dependency>
```

2、找对象

Initialization



A `RestHighLevelClient` instance needs a [REST low-level client builder](#) to be built as follows:

```
RestHighLevelClient client = new RestHighLevelClient(  
    RestClient.builder(  
        new HttpHost("localhost", 9200, "http"),  
        new HttpHost("localhost", 9201, "http")));
```

The high-level client will internally create the low-level client used to perform requests based on the provided builder. That low-level client maintains a pool of connections and starts some threads so you should close the high-level client when you are well and truly done with it and it will in turn close the internal low-level client to free those resources. This can be done through the `close`:

```
client.close();
```

In the rest of this documentation about the Java High Level Client, the `RestHighLevelClient` instance will be referenced as `client`.

3、分析这个类中的方法即可！

配置基本的项目

问题：一定要保证 我们的导入的依赖和我们的es 版本一致

默认的版本和我们使用的版本不一致


```

import org.apache.http.HttpHost;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.Credentials;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.CredentialsProvider;
import org.apache.http.impl.client.BasicCredentialsProvider;
import org.elasticsearch.client.RestClient;
import org.elasticsearch.client.RestClientBuilder;
import org.elasticsearch.client.RestHighLevelClient;

import org.springframework.beans.factory.ObjectProvider;
import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;
import
org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.boot.context.properties.PropertyMapper;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * Elasticsearch rest client infrastructure configurations.
 *
 * @author Brian Clozel
 * @author Stephane Nicoll
 */
class RestClientConfigurations {

    @Configuration(proxyBeanMethods = false)
    static class RestClientBuilderConfiguration {
        // RestClientBuilder
        @Bean
        @ConditionalOnMissingBean
        RestClientBuilder elasticsearchRestClientBuilder(RestClientProperties
properties,
            ObjectProvider<RestClientBuilderCustomizer> builderCustomizers) {
            HttpHost[] hosts =
properties.getUris().stream().map(HttpHost::create).toArray(HttpHost[]::new);
            RestClientBuilder builder = RestClient.builder(hosts);
            PropertyMapper map = PropertyMapper.get();
            map.from(properties::getUsername).whenHasText().to((username) -> {
                CredentialsProvider credentialsProvider = new
BasicCredentialsProvider();
                Credentials credentials = new
UsernamePasswordCredentials(properties.getUsername(),
                    properties.getPassword());
                credentialsProvider.setCredentials(AuthScope.ANY, credentials);
                builder.setHttpClientConfigCallback(
                    (httpClientBuilder) ->
httpClientBuilder.setDefaultCredentialsProvider(credentialsProvider));
            });
            builder.setRequestConfigCallback((requestConfigBuilder) -> {

                map.from(properties::getConnectionTimeout).whenNonNull().asInt(Duration::toMillis)
                    .to(requestConfigBuilder::setConnectTimeout);

                map.from(properties::getReadTimeout).whenNonNull().asInt(Duration::toMillis)
                    .to(requestConfigBuilder::setSocketTimeout);
            });
        }
    }
}

```

```

        return requestConfigBuilder;
    });
    builderCustomizers.orderedStream().forEach((customizer) ->
customizer.customize(builder));
    return builder;
}

}

@Configuration(proxyBeanMethods = false)
@ConditionalOnClass(RestHighLevelClient.class)
static class RestHighLevelClientConfiguration {
    // RestHighLevelClient 高级客户端，也是我们这里要讲，后面项目会用到的客户端
    @Bean
    @ConditionalOnMissingBean
    RestHighLevelClient elasticsearchRestHighLevelClient(RestClientBuilder
restClientBuilder) {
        return new RestHighLevelClient(restClientBuilder);
    }

    @Bean
    @ConditionalOnMissingBean
    RestClient elasticsearchRestClient(RestClientBuilder builder,
        ObjectProvider<RestHighLevelClient> restHighLevelClient) {
        RestHighLevelClient client = restHighLevelClient.getIfUnique();
        if (client != null) {
            return client.getLowLevelClient();
        }
        return builder.build();
    }
}

@Configuration(proxyBeanMethods = false)
static class RestClientFallbackConfiguration {
    // RestClient 普通的客户端！
    @Bean
    @ConditionalOnMissingBean
    RestClient elasticsearchRestClient(RestClientBuilder builder) {
        return builder.build();
    }
}
}
}

```

具体的Api测试！

- 1、创建索引
- 2、判断索引是否存在
- 3、删除索引
- 4、创建文档
- 5、crud文档！

```

package com.kuang;

import com.alibaba.fastjson.JSON;
import com.kuang.pojo.User;
import com.kuang.utils.EsConst;
import com.sun.corba.se.spi.activation.ServerAlreadyRegistered;
import lombok.SneakyThrows;
import org.elasticsearch.action.admin.indices.delete.DeleteIndexRequest;
import org.elasticsearch.action.bulk.BulkRequest;
import org.elasticsearch.action.bulk.BulkResponse;
import org.elasticsearch.action.delete.DeleteRequest;
import org.elasticsearch.action.delete.DeleteResponse;
import org.elasticsearch.action.get.GetRequest;
import org.elasticsearch.action.get.GetResponse;
import org.elasticsearch.action.index.IndexRequest;
import org.elasticsearch.action.index.IndexResponse;
import org.elasticsearch.action.search.SearchRequest;
import org.elasticsearch.action.search.SearchResponse;
import org.elasticsearch.action.support.master.AcknowledgedResponse;
import org.elasticsearch.action.update.UpdateRequest;
import org.elasticsearch.action.update.UpdateResponse;
import org.elasticsearch.client.RequestOptions;
import org.elasticsearch.client.RestHighLevelClient;
import org.elasticsearch.client.indices.CreateIndexRequest;
import org.elasticsearch.client.indices.CreateIndexResponse;
import org.elasticsearch.client.indices.GetIndexRequest;
import org.elasticsearch.common.unit.TimeValue;
import org.elasticsearch.common.xcontent.XContentType;
import org.elasticsearch.index.query.MatchAllQueryBuilder;
import org.elasticsearch.index.query.QueryBuilder;
import org.elasticsearch.index.query.QueryBuilders;
import org.elasticsearch.index.query.TermQueryBuilder;
import org.elasticsearch.search.SearchHit;
import org.elasticsearch.search.SearchPhase;
import org.elasticsearch.search.builder.SearchSourceBuilder;
import org.elasticsearch.search.fetch.subphase.FetchSourceContext;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.reflect.ReflectionFactory;

import java.io.IOException;
import java.util.ArrayList;
import java.util.concurrent.TimeUnit;
import java.util.function.Predicate;

/**
 * 狂神讲解 es7.6.x 高级客户端测试 API
 */
@SpringBootTest
class KuangshenEsApiApplicationTests {

    // 面向对象来操作
    @Autowired
    @Qualifier("restHighLevelClient")
    private RestHighLevelClient client;

```

```

// 测试索引的创建 Request PUT kuang_index
@Test
void testCreateIndex() throws IOException {
    // 1、创建索引请求
    CreateIndexRequest request = new CreateIndexRequest("kuang_index");
    // 2、客户端执行请求 IndicesClient, 请求后获得响应
    CreateIndexResponse createIndexResponse =
        client.indices().create(request, RequestOptions.DEFAULT);

    System.out.println(createIndexResponse);
}

// 测试获取索引, 判断其是否存在
@Test
void testExistIndex() throws IOException {
    GetIndexRequest request = new GetIndexRequest("kuang_index2");
    boolean exists = client.indices().exists(request,
RequestOptions.DEFAULT);
    System.out.println(exists);
}

// 测试删除索引
@Test
void testDeleteIndex() throws IOException {
    DeleteIndexRequest request = new DeleteIndexRequest("kuang_index");
    // 删除
    AcknowledgedResponse delete = client.indices().delete(request,
RequestOptions.DEFAULT);
    System.out.println(delete.isAcknowledged());
}

// 测试添加文档
@Test
void testAddDocument() throws IOException {
    // 创建对象
    User user = new User("狂神说", 3);
    // 创建请求
    IndexRequest request = new IndexRequest("kuang_index");

    // 规则 put /kuang_index/_doc/1
    request.id("1");
    request.timeout(TimeValue.timeValueSeconds(1));
    request.timeout("1s");

    // 将我们的数据放入请求 json
    request.source(JSON.toJSONString(user), XContentType.JSON);

    // 客户端发送请求, 获取响应的结果
    IndexResponse indexResponse = client.index(request,
RequestOptions.DEFAULT);

    System.out.println(indexResponse.toString()); //
    System.out.println(indexResponse.status()); // 对应我们命令返回的状态
    CREATED
}

// 获取文档, 判断是否存在 get /index/doc/1
@Test

```

```

void testIsExists() throws IOException {
    GetRequest getRequest = new GetRequest("kuang_index", "1");
    // 不获取返回的 _source 的上下文了
    getRequest.fetchSourceContext(new FetchSourceContext(false));
    getRequest.storedFields("_none_");

    boolean exists = client.exists(getRequest, RequestOptions.DEFAULT);
    System.out.println(exists);
}

// 获得文档的信息
@Test
void testGetDocument() throws IOException {
    GetRequest getRequest = new GetRequest("kuang_index", "1");
    GetResponse getResponse = client.get(getRequest,
RequestOptions.DEFAULT);
    System.out.println(getResponse.getSourceAsString()); // 打印文档的内容
    System.out.println(getResponse); // 返回的全部内容和命令式一样的
}

// 更新文档的信息
@Test
void testUpdateRequest() throws IOException {
    UpdateRequest updateRequest = new UpdateRequest("kuang_index","1");
    updateRequest.timeout("1s");

    User user = new User("狂神说Java", 18);
    updateRequest.doc(JSON.toJSONString(user),XContentType.JSON);

    UpdateResponse updateResponse = client.update(updateRequest,
RequestOptions.DEFAULT);
    System.out.println(updateResponse.status());
}

// 删除文档记录
@Test
void testDeleteRequest() throws IOException {
    DeleteRequest request = new DeleteRequest("kuang_index","1");
    request.timeout("1s");

    DeleteResponse deleteResponse = client.delete(request,
RequestOptions.DEFAULT);
    System.out.println(deleteResponse.status());
}

// 特殊的，真的项目一般都会批量插入数据！
@Test
void testBulkRequest() throws IOException {
    BulkRequest bulkRequest = new BulkRequest();
    bulkRequest.timeout("10s");

    ArrayList<User> userList = new ArrayList<>();
    userList.add(new User("kuangshen1",3));
    userList.add(new User("kuangshen2",3));
    userList.add(new User("kuangshen3",3));
    userList.add(new User("qinjiang1",3));
    userList.add(new User("qinjiang1",3));
    userList.add(new User("qinjiang1",3));
}

```

```

        // 批处理请求
        for (int i = 0; i < userList.size() ; i++) {
            // 批量更新和批量删除，就在这里修改对应的请求就可以了
            bulkRequest.add(
                new IndexRequest("kuang_index")
                    .id(""+(i+1))

            .source(JSON.toJSONString(userList.get(i)),XContentType.JSON));
        }

        BulkResponse bulkResponse = client.bulk(bulkRequest,
RequestOptions.DEFAULT);
        System.out.println(bulkResponse.hasFailures()); // 是否失败，返回 false 代表
成功!
    }

    // 查询
    // SearchRequest 搜索请求
    // SearchSourceBuilder 条件构造
    // HighlightBuilder 构建高亮
    // TermQueryBuilder 精确查询
    // MatchAllQueryBuilder
    // xxx QueryBuilder 对应我们刚才看到的命令!

    @Test
    void testSearch() throws IOException {
        SearchRequest searchRequest = new SearchRequest("kuang_index");
        // 构建搜索条件
        SearchSourceBuilder sourceBuilder = new SearchSourceBuilder();
        sourceBuilder.highlighter()
        // 查询条件，我们可以使用 QueryBuilders 工具来实现
        // QueryBuilders.termQuery 精确
        // QueryBuilders.matchAllQuery() 匹配所有
        TermQueryBuilder termQueryBuilder = QueryBuilders.termQuery("name",
"qinjiang1");
        //      MatchAllQueryBuilder matchAllQueryBuilder =
QueryBuilders.matchAllQuery();
        sourceBuilder.query(termQueryBuilder);
        sourceBuilder.timeout(new TimeValue(60,TimeUnit.SECONDS));

        searchRequest.source(sourceBuilder);

        SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);
        System.out.println(JSON.toJSONString(searchResponse.getHits()));
        System.out.println("=====");
        for (SearchHit documentFields : searchResponse.getHits().getHits()) {
            System.out.println(documentFields.getSourceAsMap());
        }
    }
}

```

实战

最终的效果！



爬虫

数据问题？数据库获取，消息队列中获取中，都可以成为数据源，爬虫！

爬取数据：（获取请求返回的页面信息，筛选出我们想要的的数据就可以了！）

前后端分离

搜索高亮

以后大家学习完我这个es就可以编写基本es业务了！

小结

资料获取：公众号关注狂神说，回复 es 获取资料!