

主讲老师：Fox

ES版本：v7.17.3

ES环境搭建视频：<https://pan.baidu.com/s/1PsTNbpDy--M-pvFWb3aehQ?pwd=nwxi>

- 1 文档：[1.ElasticSearch快速入门实战.note](#)
- 2 链接：<http://note.youdao.com/noteshare?id=d5d5718ae542f274ba0fda4284a53231&sub=68E590656C7A48858C7F6997D4A1511A>

全文检索

什么是全文检索

倒排索引

ElasticSearch简介

ElasticSearch是什么

起源——Lucene

Elasticsearch的诞生

ElasticSearch版本特性

ElasticSearch vs Solr

Elastic Stack介绍

ElasticSearch应用场景

ElasticSearch快速开始

ElasticSearch安装运行

环境准备

下载并解压ElasticSearch

修改JVM配置

启动ElasticSearch服务

客户端Kibana安装

Elasticsearch安装分词插件

ElasticSearch基本概念

关系型数据库 VS ElasticSearch

索引 (Index)

文档 (Document)

ElasticSearch索引操作

创建索引

查询索引

删除索引

ElasticSearch文档操作

添加 (索引) 文档

修改文档

并发场景下修改文档

查询文档

删除文档

ElasticSearch文档批量操作

批量写入

批量读取

全文检索

数据分类：

- 结构化数据：固定格式，有限长度 比如mysql存的数据
- 非结构化数据：不定长，无固定格式 比如邮件，word文档，日志

- 半结构化数据：前两者结合 比如xml, html

搜索分类：

- 结构化数据搜索： 使用关系型数据库
- 非结构化数据搜索
 - 顺序扫描
 - 全文检索

设想一个关于搜索的场景，假设我们要搜索一首诗句内容中带“前”字的古诗

name	content	author
静夜思	床前明月光,疑是地上霜。举头望明月, 低头思故乡。	李白
望庐山瀑布	日照香炉生紫烟, 遥看瀑布挂前川。飞流直下三千尺,疑是银河落九天。	李白
...

思考：用传统关系型数据库和ES 实现会有什么差别？

如果用像 MySQL 这样的 RDBMS 来存储古诗的话，我们应该会去使用这样的 SQL 去查询

```
1 select name from poems where content like "%前%"
```

这种我们称为顺序扫描法，需要遍历所有的记录进行匹配。不但效率低，而且不符合我们搜索时的期望，比如我们在搜索“ABCD”这样的关键词时，通常还希望看到“A”,“AB”,“CD”, “ABC” 的搜索结果。

什么是全文检索

全文检索是指：

- 通过一个程序扫描文本中的每一个单词，针对单词建立索引，并保存该单词在文本中的位置、以及出现的次数
- 用户查询时，通过之前建立好的索引来查询，将索引中单词对应的文本位置、出现的次数返回给用户，因为有了具体文本的位置，所以就可以将具体内容读取出来了

全站 博客 下载 代码 用户 课程 专栏 问答 商城 更多

综合 最新 热门 ☐ VIP内容

相关结果约21,970个

ElasticSearch最新版快速入门详解

本文把最新版的ElasticSearch和kibana的知识点用通俗易懂的语言来展现，并会在核心概念上和MySQL对比，结合示例进行图文并茂的讲解，同时还给大家提供百分百成功的极速安装配置方法哦！

3.6万+

261

122

且听_风吟

2020-05-29

**教你快速入门ElasticSearch，超详细简单~**

教你快速入门ElasticSearch，超详细简单~ 一. 初探ElasticSearch 1.1 什么是ElasticSearch? ElasticSearch，简称为ES，它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；它的扩展性很...

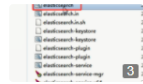
3721

7

5

暗余

2021-06-17

**Elasticsearch通关教程（一）：基础入门**

Elasticsearch是一个高度可扩展的、开源的、基于Lucene的全文搜索和分析引擎。它允许您快速、近实时地存储，搜索和分析大量数据，并支持多租户。Elasticsearch也使用Java开发并使用Lucene作为其核心来实现所有...

2.5万+

17

7

weixin_33806... 2019-03-19

ElasticSearch快速入门

目录1 初识ElasticSearch1.1 基于数据库查询的问题1.2 倒排索引1.3 ES存储和查询的原理1.4 ES概念详解1.5 知识总结2 安装ElasticSearch2.1 ES安装2.2 ES辅助工具安装3 ElasticSearch核心概念4 脚本操作ES4.1 RESTful...

916

1

赵广陆

2020-12-22

**ElasticSearch快速学习---30分钟入门ElasticSearch**

ElasticSearch快速学习 ElasticSearch原理，30分钟入门ElasticSearch 目录 1 解析es的分布式架构 2 分片和副本机制 3 单节点环境下创建索引分析 4 两个节点

搜索原理简单概括的话可以分为这么几步：

- 内容爬取，停顿词过滤比如一些无用的像"的"，"了"之类的语气词/连接词
- 内容分词，提取关键词
- 根据关键词建立倒排索引
- 用户输入关键词进行搜索

倒排索引

索引就类似于目录，平时我们使用的都是索引，都是通过主键定位到某条数据，那么倒排索引呢，刚好相反，数据对应到主键。



这里以一个博客文章的内容为例:

正排索引（正向索引）

文章ID	文章标题	文章内容
1	浅析JAVA设计模式	JAVA设计模式是每一个JAVA程序员都应该掌握的进阶知识
2	JAVA多线程设计模式	JAVA多线程与设计模式结合

倒排索引（反向索引）

假如，我们有一个站内搜索的功能，通过某个关键词来搜索相关的文章，那么这个关键词可能出现在标题中，也可能出现在文章内容中，那我们将会在创建或修改文章的时候，建立一个关键词与文章的对应关系表，这种，我们可以称之为倒排索引。

like %java设计模式% java 设计模式

关键词	文章ID
JAVA	1
设计模式	1,2
多线程	2

简单理解，正向索引是通过key找value，反向索引则是通过value找key。ES底层在检索时底层使用的就是倒排索引。

ElasticSearch简介

ElasticSearch是什么

ElasticSearch（简称ES）是一个分布式、RESTful 风格的搜索和数据分析引擎，是用Java开发并且是当前最流行的开源的企业级搜索引擎，能够达到近实时搜索，稳定，可靠，快速，安装使用方便。

客户端支持Java、.NET（C#）、PHP、Python、Ruby等多种语言。

官方网站: <https://www.elastic.co/>

下载地址: <https://www.elastic.co/cn/downloads/past-releases#elasticsearch>

搜索引擎排名：

☐ include secondary database models

26 systems in ranking, May 2022

Rank			DBMS	Database Model	Score		
May 2022	Apr 2022	May 2021			May 2022	Apr 2022	May 2021
1.	1.	1.	Elasticsearch	Search engine, Multi-model	157.69	-3.14	+2.34
2.	2.	2.	Splunk	Search engine	96.35	+1.11	+4.24
3.	3.	3.	Solr	Search engine, Multi-model	57.26	-0.48	+6.07
4.	4.	4.	MarkLogic	Multi-model	9.85	-0.21	+0.33
5.	5.	5.	Algolia	Search engine	8.08	-0.29	+0.36
6.	6.	↑ 7.	Microsoft Azure Search	Search engine	7.54	-0.61	+1.49
7.	7.	↓ 6.	Sphinx	Search engine	6.72	-0.53	-0.86

参考网站: <https://db-engines.com/en/ranking/search+engine>

起源——Lucene

- 基于Java语言开发的搜索引擎库类
- 创建于1999年，2005年成为Apache 顶级开源项目
- Lucene具有高性能、易扩展的优点
- Lucene的局限性：
 - 只能基于Java语言开发
 - 类库的接口学习曲线陡峭
 - 原生并不支持水平扩展

Elasticsearch的诞生

Elasticsearch是构建在Apache Lucene之上的开源分布式搜索引擎。

- 2004年 Shay Banon 基于Lucene开发了Compass
- 2010年 Shay Banon重写了Compass，取名Elasticsearch
 - 支持分布式，可水平扩展
 - 降低全文检索的学习曲线，可以被任何编程语言调用

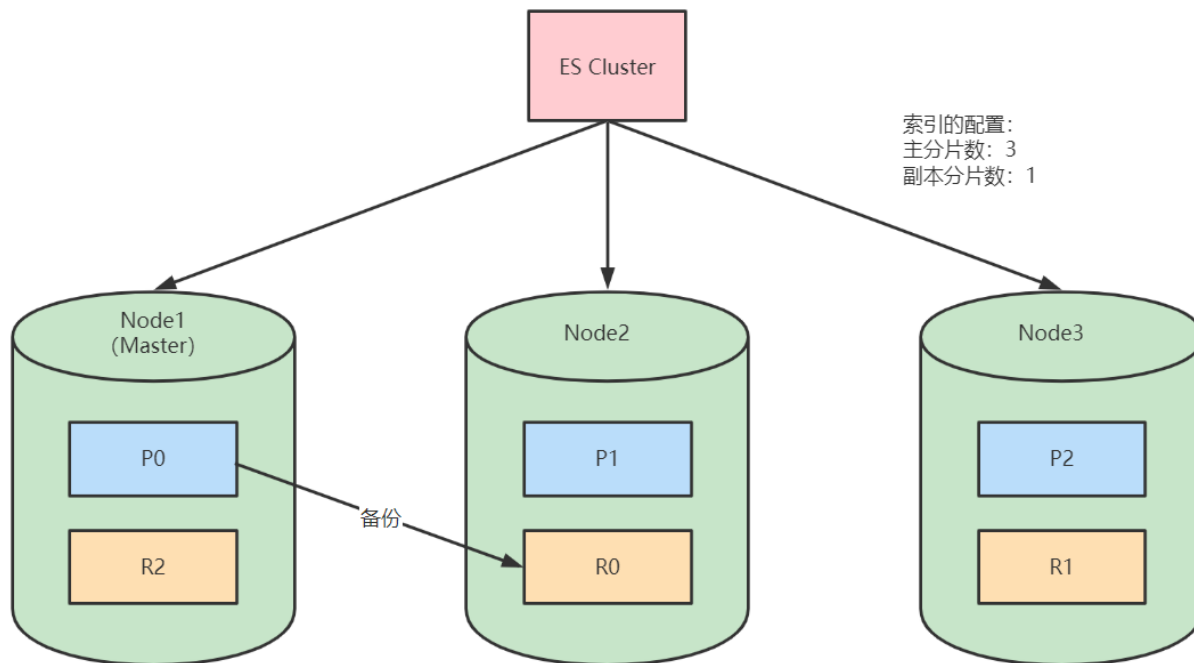


Elasticsearch 与 Lucene 核心库竞争的优势在于：

- 完美封装了 Lucene 核心库，设计了友好的 Restful-API，开发者无需过多关注底层机制，直接开箱即用。
- 分片与副本机制，直接解决了集群下性能与高可用问题。

ES Server进程 3节点 raft (奇数节点)

数据分片 -》lucene实例 分片和副本数 1个ES节点可以有多个lucene实例。也可以指定一个索引的多个分片



ElasticSearch版本特性

5.x新特性

- Lucene 6.x, 性能提升，默认打分机制从TF-IDF改为BM 25
- 支持Ingest节点/ Painless Scripting / Completion suggested支持/原生的Java REST客户端
- Type标记成deprecated，支持了Keyword的类型
- 性能优化
 - 内部引擎移除了避免同一文档并发更新的竞争锁，带来15% - 20%的性能提升
 - Instant aggregation,支持分片，上聚合的缓存
 - 新增了Profile API

6.x新特性

- Lucene 7.x
- 新功能

- 跨集群复制(CCR)
 - 索引生命周期管理
 - SQL的支持
- 更友好的升级及数据迁移
 - 在主要版本之间的迁移更为简化，体验升级
 - 全新的基于操作的数据复制框架，可加快恢复数据
- 性能优化
 - 有效存储稀疏字段的新方法，降低了存储成本
 - 在索引时进行排序，可加快排序的查询性能

7.x新特性

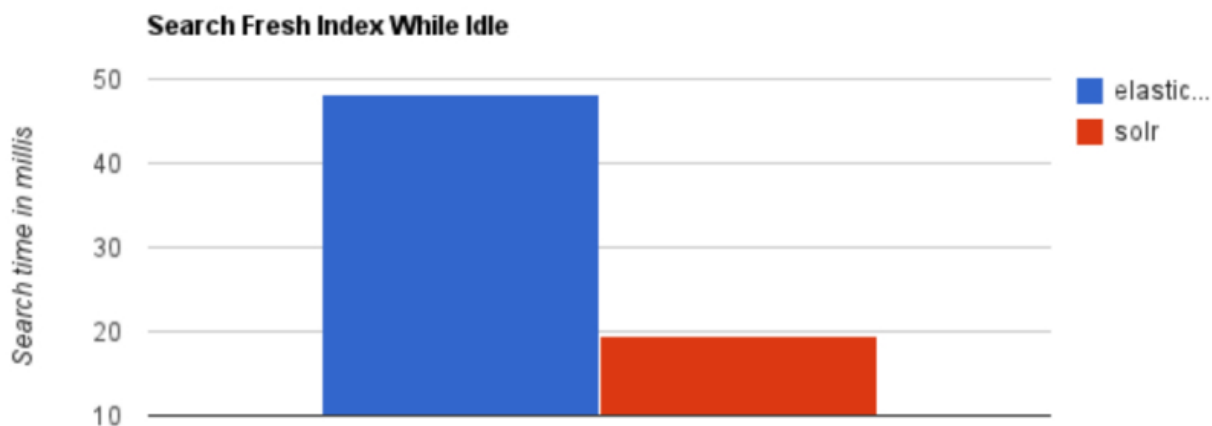
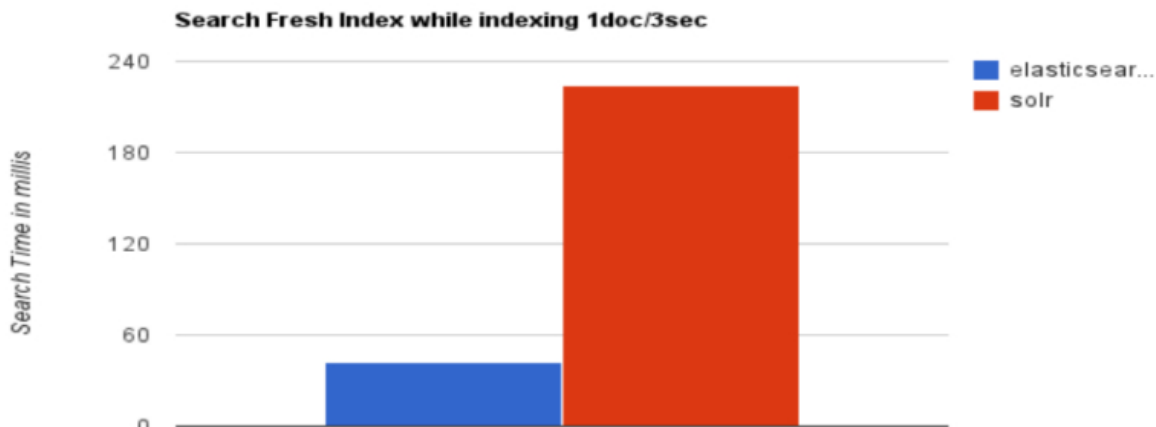
- Lucene 8.0
- 重大改进-正式废除单个索引下多Type的支持
- 7.1开始，Security 功能免费使用
- ECK - Elasticsearch Operator on Kubernetes
- 新功能
 - New Cluster coordination
 - Feature——Complete High Level REST Client
 - Script Score Query
- 性能优化
 - 默认的Primary Shard数从5改为1,避免Over Sharding
 - 性能优化， 更快的Top K

8.x新特性

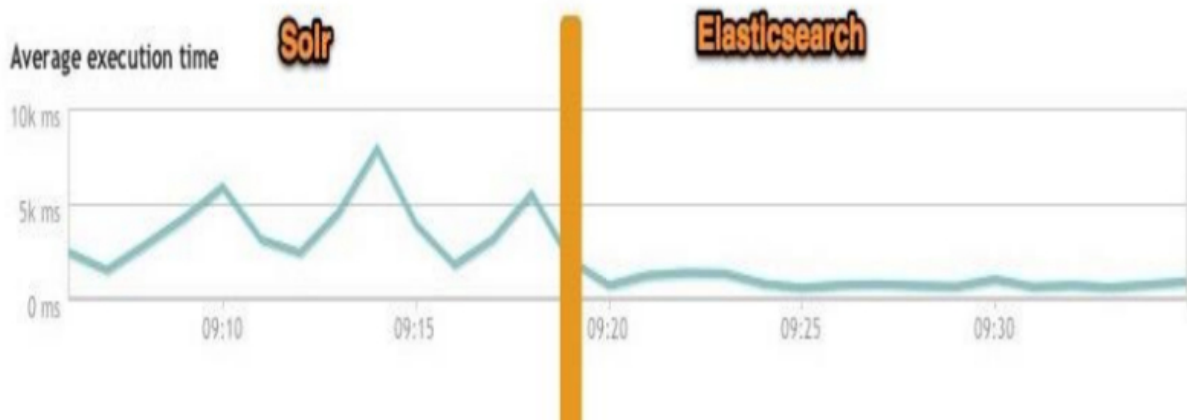
- Rest API相比较7.x而言做了比较大的改动（比如彻底删除_type）
- 默认开启安全配置
- 存储空间优化：对倒排文件使用新的编码集，对于keyword、match_only_text、text类型字段有效，有3.5%的空间优化提升，对于新建索引和segment自动生效。
- 优化geo_point, geo_shape类型的索引（写入）效率：15%的提升。
- 技术预览版KNN API发布，（K邻近算法），跟推荐系统、自然语言排名相关。
- <https://www.elastic.co/guide/en/elastic-stack/current/elasticsearch-breaking-changes.html>

ElasticSearch vs Solr

Solr 是第一个基于 Lucene 核心库功能完备的搜索引擎产品，诞生远早于 Elasticsearch。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。



大型互联网公司，实际生产环境测试，将搜索引擎从Solr转到 Elasticsearch以后的平均查询速度有了50倍的提升。



总结：

- 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，比如JSON、XML、CSV，而 Elasticsearch 仅支持 json文件格式。
- 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。
- 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。

Elastic Stack介绍

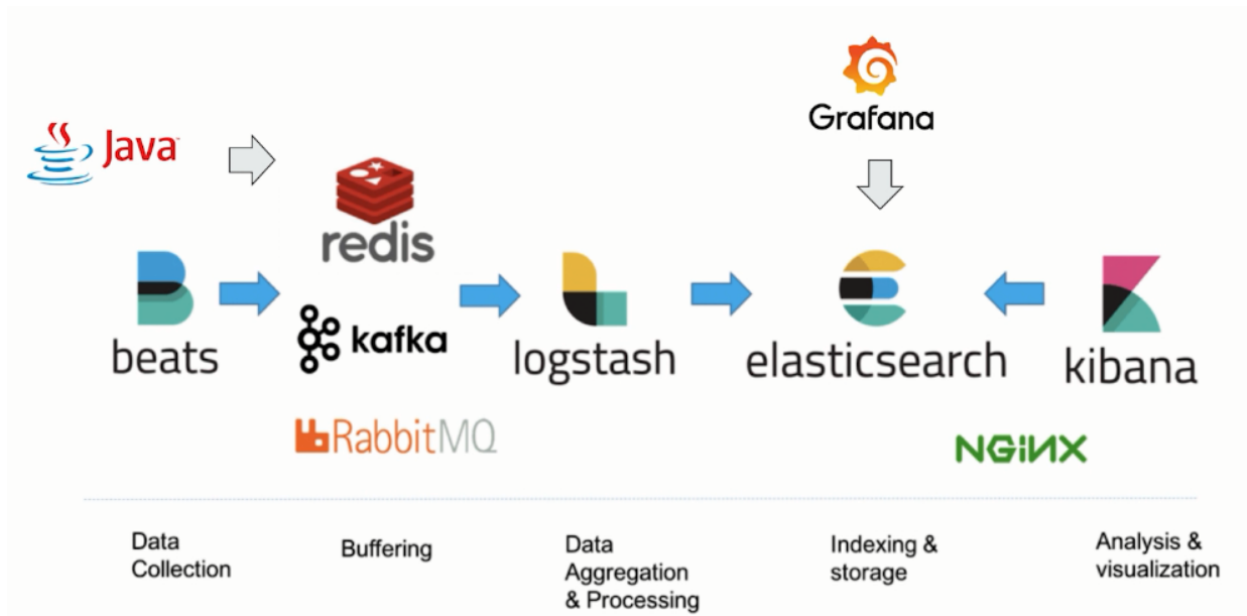
在Elastic Stack之前我们听说过ELK，ELK分别是Elasticsearch，Logstash，Kibana这三款软件在一起的简称，在发展的过程中又有新的成员Beats的加入，就形成了Elastic Stack。



Elastic Stack生态圈

在Elastic Stack生态圈中Elasticsearch作为数据存储和搜索，是生态圈的基石，Kibana在上层提供用户一个可视化及操作的界面，Logstash和Beat可以对数据进行收集。在上图的右侧X-Pack部分则是Elastic公司提供的商业项目。

指标分析/日志分析：

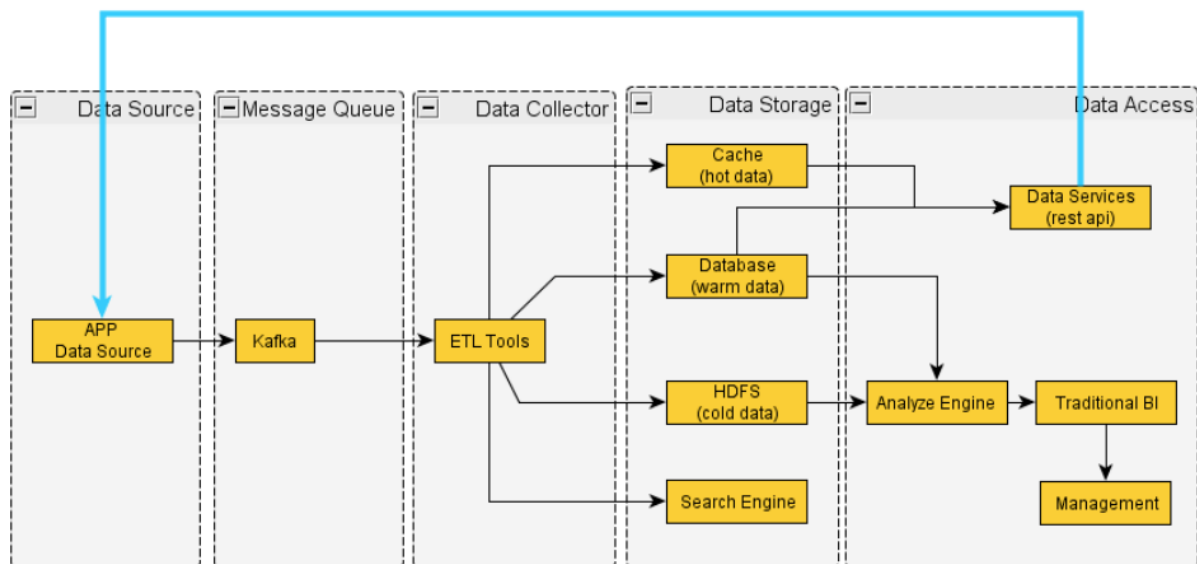


ElasticSearch应用场景

- 站内搜索
- 日志管理与分析
- 大数据分析
- 应用性能监控
- 机器学习

国内现在有大量的公司都在使用 Elasticsearch，包括携程、滴滴、今日头条、饿了么、360安全、小米、vivo等诸多知名公司。除了搜索之外，结合Kibana、Logstash、Beats，Elastic Stack还被广泛运用在大数据近实时分析领域，包括日志分析、指标监控、信息安全等多个领域。它可以帮助你探索海量结构化、非结构化数据，按需创建可视化报表，对监控数据设置报警阈值，甚至通过使用机器学习技术，自动识别异常状况。

通用数据处理流程：



ElasticSearch快速开始

ElasticSearch安装运行

环境准备

- 运行Elasticsearch，需安装并配置JDK
 - 设置\$JAVA_HOME
- 各个版本对Java的依赖 https://www.elastic.co/support/matrix#matrix_jvm
 - Elasticsearch 5需要Java 8以上的版本
 - Elasticsearch 从6.5开始支持Java 11
 - 7.0开始，内置了Java环境
- ES比较耗内存，建议虚拟机4G或以上内存，jvm1g以上的内存分配

可以参考es的环境文件elasticsearch-env.bat

```

if defined ES_JAVA_HOME (
    set JAVA="%ES_JAVA_HOME%\bin\java.exe"
    set JAVA_TYPE=ES_JAVA_HOME
) else if defined JAVA_HOME (
    rem fallback to JAVA_HOME
    echo "warning: usage of JAVA_HOME is deprecated, use ES_JAVA_HOME" >&2
    set JAVA="%JAVA_HOME%\bin\java.exe"
    set "ES_JAVA_HOME=%JAVA_HOME%"
    set JAVA_TYPE=JAVA_HOME
) else (
    rem use the bundled JDK (default)
    set JAVA="%ES_HOME%\jdk\bin\java.exe"
    set "ES_JAVA_HOME=%ES_HOME%\jdk"
    set JAVA_TYPE=bundled JDK
)
  
```

ES的jdk环境生效的优先级配置ES_JAVA_HOME>JAVA_HOME>ES_HOME

下载并解压ElasticSearch

下载地址：<https://www.elastic.co/cn/downloads/past-releases#elasticsearch>

选择版本：7.17.3

Elasticsearch

7.17.3

Elasticsearch 7.17.3

April 21, 2022

[See Release Notes](#)

[Download](#)

ElasticSearch文件目录结构

目录	描述
bin	脚本文件，包括启动elasticsearch，安装插件，运行统计数据等
config	配置文件目录，如elasticsearch配置、角色配置、jvm配置等。
jdk	java运行环境
data	默认的数据存放目录，包含节点、分片、索引、文档的所有数据，生产环境需要修改。
lib	elasticsearch依赖的Java类库
logs	默认的日志文件存储路径，生产环境需要修改。
modules	包含所有的Elasticsearch模块，如Cluster、Discovery、Indices等。
plugins	已安装插件目录

主配置文件elasticsearch.yml

- cluster.name

当前节点所属集群名称，多个节点如果要组成同一个集群，那么集群名称一定要配置成相同。默认值elasticsearch，生产环境建议根据ES集群的使用目的修改成合适的名字。

- node.name

当前节点名称，默认值当前节点部署所在机器的主机名，所以如果一台机器上要起多个ES节点的话，需要通过配置该属性明确指定不同的节点名称。

- path.data

配置数据存储目录，比如索引数据等，默认值 \$ES_HOME/data，生产环境下强烈建议部署到另外的安全目录，防止ES升级导致数据被误删除。

- path.logs

配置日志存储目录，比如运行日志和集群健康信息等，默认值 `$ES_HOME/logs`，生产环境下强烈建议部署到另外的安全目录，防止ES升级导致数据被误删除。

- bootstrap.memory_lock

配置ES启动时是否进行内存锁定检查，默认值true。

ES对于内存的需求比较大，一般生产环境建议配置大内存，如果内存不足，容易导致内存交换到磁盘，严重影响ES的性能。所以默认启动时进行相应大小内存的锁定，如果无法锁定则会启动失败。

非生产环境可能机器内存本身就很小，能够供给ES使用的就更小，如果该参数配置为true的话很可能导致无法锁定内存以致ES无法成功启动，此时可以修改为false。

- network.host

配置能够访问当前节点的主机，默认值为当前节点所在机器的本机回环地址127.0.0.1 和 `[::1]`，这就导致默认情况下只能通过当前节点所在主机访问当前节点。可以配置为 `0.0.0.0`，表示所有主机均可访问。

- http.port

配置当前ES节点对外提供服务的http端口，默认值 9200

- discovery.seed_hosts

配置参与集群节点发现过程的主机列表，说白了就是集群中所有节点所在的主机列表，可以是具体的IP地址，也可以是可解析的域名。

- cluster.initial_master_nodes

配置ES集群初始化时参与master选举的节点名称列表，必须与node.name配置的一致。ES集群首次构建完成后，应该将集群中所有节点的配置文件中的 `cluster.initial_master_nodes`配置项移除，重启集群或者将新节点加入某个已存在的集群时切记不要设置该配置项。

```
1 #ES开启远程访问
2 network.host: 0.0.0.0
```

修改JVM配置

修改config/jvm.options配置文件，调整jvm堆内存大小

```
1 vim jvm.options
2 -Xms4g
3 -Xmx4g
```

配置的建议

- Xms和Xmx设置成一样
- Xmx不要超过机器内存的50%

- 不要超过30GB - <https://www.elastic.co/cn/blog/a-heap-of-trouble>

启动ElasticSearch服务

Windows

直接运行elasticsearch.bat

Linux (centos7)

ES不允许使用root账号启动服务，如果你当前账号是root，则需要创建一个专有账户

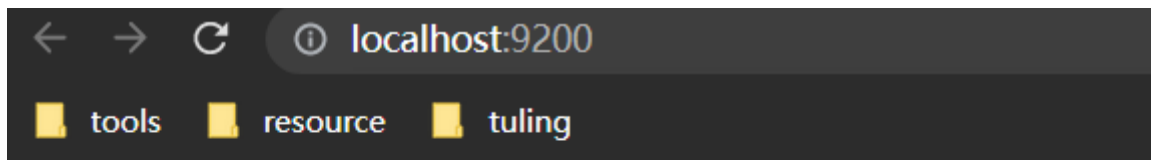
```
1 #非root用户
2 bin/elasticsearch
3
4 # -d 后台启动
5 bin/elasticsearch -d
```

```
[2022-05-29T16:05:25,654][ERROR][o.e.b.ElasticsearchUncaughtExceptionHandler] [redis] uncaught exception in thread [main]
org.elasticsearch.bootstrap.StartupException: java.lang.RuntimeException: can not run elasticsearch as root
    at org.elasticsearch.bootstrap.Elasticsearch.init(Elasticsearch.java:170) ~[elasticsearch-7.17.3.jar:7.17.3]
    at org.elasticsearch.bootstrap.Elasticsearch.execute(Elasticsearch.java:157) ~[elasticsearch-7.17.3.jar:7.17.3]
    at org.elasticsearch.cli.EnvironmentAwareCommand.execute(EnvironmentAwareCommand.java:77) ~[elasticsearch-7.17.3.jar:7.17.3]
```

注意：es默认不能用root用户启动，生产环境建议为elasticsearch创建用户。

```
1 #为elasticsearch创建用户并赋予相应权限
2 adduser es
3 passwd es
4 chown -R es:es .
5 elasticsearch-17.3
```

运行<http://localhost:9200/>



```
{
  name: "LAPTOP-UTD9471P",
  cluster_name: "elasticsearch",
  cluster_uuid: "DY1hUA3YTT6SGILETjddeA",
  - version: {
    number: "7.17.3",
    build_flavor: "default",
    build_type: "zip",
    build_hash: "5ad023604c8d7416c9eb6c0eadb62b14e766caff",
    build_date: "2022-04-19T08:11:19.070913226Z",
    build_snapshot: false,
    lucene_version: "8.11.1",
    minimum_wire_compatibility_version: "6.8.0",
    minimum_index_compatibility_version: "6.0.0-beta1"
  },
  tagline: "You Know, for Search"
}
```

如果ES服务启动异常，会有提示：

```
ERROR: [2] bootstrap checks failed. You must address the points described in the following [2] lines before starting Elasticsearch.
bootstrap check failure [1] of [2]: max virtual memory areas vm.max_map_count [65536] is too low, increase to at least [262144]
bootstrap check failure [2] of [2]: the default discovery settings are unsuitable for production use; at least one of [discovery.seed_hosts, discovery
.seed_providers, cluster.initial_master_nodes] must be configured
ERROR: Elasticsearch did not exit normally - check the logs at /home/es/elasticsearch-7.17.3/logs/elasticsearch.log
[2022-05-22T14:23:04.497][INFO ][o.e.n.Node               ] [hadoop01] stopping ...
[2022-05-22T14:23:04.567][INFO ][o.e.n.Node               ] [hadoop01] stopped
[2022-05-22T14:23:04.569][INFO ][o.e.n.Node               ] [hadoop01] closing ...
[2022-05-22T14:23:04.610][INFO ][o.e.n.Node               ] [hadoop01] closed
```

启动ES服务常见错误解决方案

[1]: max file descriptors [4096] for elasticsearch process is too low, increase to at least [65536]

ES因为需要大量的创建索引文件，需要大量的打开系统的文件，所以我们需要解除linux系统当中打开文件最大数目的限制，不然ES启动就会抛错

```
1 #切换到root用户
2 vim /etc/security/limits.conf
3
4 末尾添加如下配置：
5 * soft nfile 65536
6 * hard nfile 65536
7 * soft nproc 4096
8 * hard nproc 4096
```

[2]: max number of threads [1024] for user [es] is too low, increase to at least [4096]

无法创建本地线程问题,用户最大可创建线程数太小

```
1 vim /etc/security/limits.d/20-nproc.conf
2
3 改为如下配置:
4 * soft nproc 4096
```

[3]: max virtual memory areas vm.max_map_count [65530] is too low, increase to at least [262144]

最大虚拟内存太小,调大系统的虚拟内存

```
1 vim /etc/sysctl.conf
2 追加以下内容:
3 vm.max_map_count=262144
4 保存退出之后执行如下命令:
5 sysctl -p
```

[4]: the default discovery settings are unsuitable for production use; at least one of [discovery.seed_hosts, discovery.seed_providers, cluster.initial_master_nodes] must be configured

缺少默认配置, 至少需要配置

discovery.seed_hosts/discovery.seed_providers/cluster.initial_master_nodes中的一个参数.

- discovery.seed_hosts: 集群主机列表
- discovery.seed_providers: 基于配置文件配置集群主机列表
- cluster.initial_master_nodes: 启动时初始化的参与选主的node, 生产环境必填

```
1 vim config/elasticsearch.yml
2 #添加配置
3 discovery.seed_hosts: ["127.0.0.1"]
4 cluster.initial_master_nodes: ["node-1"]
5
6 #或者 单节点 (集群单节点)
7 discovery.type: node-single
```

客户端Kibana安装

Kibana是一个开源分析和可视化平台, 旨在与Elasticsearch协同工作。

1) 下载并解压缩Kibana

下载地址: <https://www.elastic.co/cn/downloads/past-releases#kibana>

选择版本: 7.17.3

Kibana

7.17.3

Kibana 7.17.3

See Release Notes

Download

April 21, 2022

2) 修改Kibana.yml

```
1 vim config/kibana.yml
2
3 server.port: 5601
4 server.host: "localhost" #服务器ip
5 elasticsearch.hosts: ["http://localhost:9200"] #elasticsearch的访问地址
6 i18n.locale: "zh-CN" #Kibana汉化
```

3) 运行Kibana

注意: kibana也需要非root用户启动

```
1 bin/kibana
2 #后台启动
3 nohup bin/kibana &
```

访问Kibana: <http://localhost:5601/>

elastic

搜索 Elastic

开发工具

控制台

Search Profiler

Grok Debugger

Painless 实验室

公测版

历史记录

设置

帮助

1 #查看ES节点

2 GET /_cat/nodes

3

1 127.0.0.1 6 96 9 cdfhilmrstw * LAPTOP-UTD9471P

2

cat API

```
1 /_cat/allocation #查看单节点的shard分配整体情况
2 /_cat/shards #查看各shard的详细情况
3 /_cat/shards/{index} #查看指定分片的详细情况
4 /_cat/master #查看master节点信息
5 /_cat/nodes #查看所有节点信息
6 /_cat/indices #查看集群中所有index的详细信息
7 /_cat/indices/{index} #查看集群中指定index的详细信息
8 /_cat/segments #查看各index的segment详细信息,包括segment名, 所属shard, 内存(磁盘)占用大小, 是否刷盘
9 /_cat/segments/{index} #查看指定index的segment详细信息
10 /_cat/count #查看当前集群的doc数量
```

```
11 /_cat/count/{index} #查看指定索引的doc数量
12 /_cat/recovery #查看集群内每个shard的recovery过程.调整replica。
13 /_cat/recovery/{index} #查看指定索引shard的recovery过程
14 /_cat/health #查看集群当前状态：红、黄、绿
15 /_cat/pending_tasks #查看当前集群的pending task
16 /_cat/aliases #查看集群中所有alias信息,路由配置等
17 /_cat/aliases/{alias} #查看指定索引的alias信息
18 /_cat/thread_pool #查看集群各节点内部不同类型的threadpool的统计信息,
19 /_cat/plugins #查看集群各个节点上的plugin信息
20 /_cat/fielddata #查看当前集群各个节点的fielddata内存使用情况
21 /_cat/fielddata/{fields} #查看指定field的内存使用情况,里面传field属性对应的值
22 /_cat/nodeattrs #查看单节点的自定义属性
23 /_cat/repositories #输出集群中注册快照存储库
24 /_cat/templates #输出当前正在存在的模板信息
```

Elasticsearch安装分词插件

Elasticsearch提供插件机制对系统进行扩展

以安装analysis-icu这个分词插件为例

在线安装

```
1 #查看已安装插件
2 bin/elasticsearch-plugin list
3 #安装插件
4 bin/elasticsearch-plugin install analysis-icu
5 #删除插件
6 bin/elasticsearch-plugin remove analysis-icu
```

注意：安装和删除完插件后，需要重启ES服务才能生效。

测试分词效果

```
1 POST _analyze
2 {
3   "analyzer":"icu_analyzer",
4   "text":"中华人民共和国"
5 }
```

```
#unicode支持，中文分词良好
POST _analyze
{
  "analyzer": "icu_analyzer",
  "text": "中华人民共和国"
}

1 {
2   "tokens" : [
3     {
4       "token" : "中华",
5       "start_offset" : 0,
6       "end_offset" : 2,
7       "type" : "<IDEOGRAPHIC>",
8       "position" : 0
9     },
10    {
11      "token" : "人民",
12      "start_offset" : 2,
13      "end_offset" : 4,
14      "type" : "<IDEOGRAPHIC>",
15      "position" : 1
16    },
17    {
18      "token" : "共和国",
19      "start_offset" : 4,
20      "end_offset" : 7,
21      "type" : "<IDEOGRAPHIC>",
22      "position" : 2
23    }
24  ]
25 }
```

离线安装

本地下载相应的插件，解压，然后手动上传到elasticsearch的plugins目录，然后重启ES实例就可以了。

比如ik中文分词插件：<https://github.com/medcl/elasticsearch-analysis-ik>

测试分词效果

```
1 #ES的默认分词设置是standard，会单字拆分
2 POST _analyze
3 {
4   "analyzer": "standard",
5   "text": "中华人民共和国"
6 }
7
8 #ik_smart:会做最粗粒度的拆
9 POST _analyze
10 {
11   "analyzer": "ik_smart",
12   "text": "中华人民共和国"
13 }
14
15 #ik_max_word:会将文本做最细粒度的拆分
16 POST _analyze
17 {
18   "analyzer": "ik_max_word",
19   "text": "中华人民共和国"
```

```
20 }  
21
```

创建索引时可以指定IK分词器作为默认分词器

```
1 PUT /es_db  
2 {  
3   "settings" : {  
4     "index" : {  
5       "analysis.analyzer.default.type": "ik_max_word"  
6     }  
7   }  
8 }
```

```
{  
  "es_db" : {  
    "aliases" : { },  
    "mappings" : { },  
    "settings" : {  
      "index" : {  
        "routing" : {  
          "allocation" : {  
            "include" : {  
              "_tier_preference" : "data_content"  
            }  
          }  
        },  
        "number_of_shards" : "1",  
        "provided_name" : "es_db",  
        "creation_date" : "1653825219536",  
        "analysis" : {  
          "analyzer" : {  
            "default" : {  
              "type" : "ik_max_word"  
            }  
          }  
        },  
        "number_of_replicas" : "1",  
        "uuid" : "ICFB8e0hR6uXzq9KLbrLIA",  
        "version" : {  
          "created" : "7170399"  
        }  
      }  
    }  
  }  
}
```

ElasticSearch基本概念

关系型数据库 VS ElasticSearch

- 在7.0之前，一个 Index可以设置多个Types
- 目前Type已经被Deprecated，7.0开始，一个索引只能创建一个Type - "_doc"
- 传统关系型数据库和Elasticsearch的区别:
 - Elasticsearch- Schemaless /相关性/高性能全文检索
 - RDMS —事务性/ Join

关系型数据库	Database (数据库)	Table (表)	Row (行)	Column (列)
ElasticSearch	Index (索引库)	Type (类型)	Document (文档)	Field (字段)

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，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。

```
GET /es_db/

{
  "es_db" : {
    "aliases" : { },
    "mappings" : {
      "properties" : {
        "address" : { },
        "age" : { },
        "name" : { },
        "remark" : { },
        "sex" : { }
      }
    },
    "settings" : {
      "index" : {
        "routing" : { },
        "number_of_shards" : "1",
        "provided_name" : "es_db",
        "creation_date" : "1651740974758",
        "number_of_replicas" : "1",
        "uuid" : "Y6xCnX2XRGal2Qg6zWBKYg",
        "version" : {
          "created" : "7170399"
        }
      }
    }
  }
}
```

文档 (Document)

- Elasticsearch是面向文档的，文档是所有可搜索数据的最小单位。
 - 日志文件中的日志项
 - 一本电影的具体信息/一张唱片的详细信息
 - MP3播放器里的一首歌/一篇PDF文档中的具体内容
- 文档会被序列化成JSON格式，保存在Elasticsearch中
 - JSON对象由字段组成
 - 每个字段都有对应的字段类型(字符串/数值/布尔/日期/二进制/范围类型)
- 每个文档都有一个Unique ID
 - 可以自己指定ID或者通过Elasticsearch自动生成
- 一篇文档包含了一系列字段，类似数据库表中的一条记录
- JSON文档，格式灵活，不需要预先定义格式
 - 字段的类型可以指定或者通过Elasticsearch自动推算
 - 支持数组/支持嵌套

文档元数据

```
GET /es_db/_doc/2

2 {
3   "_index" : "es_db",
4   "_type" : "_doc",
5   "_id" : "2",
6   "_version" : 2,
7   "_seq_no" : 15,
8   "_primary_term" : 6,
9   "found" : true,
10  "_source" : {
11    "name" : "李四",
12    "sex" : 1,
13    "age" : 28,
14    "address" : "广州荔湾大厦",
15    "remark" : "java assistant"
16  }
17 }
```

元数据，用于标注文档的相关信息：

- **_index**: 文档所属的索引名
- **_type**: 文档所属的类型名
- **_id**: 文档唯一Id
- **_source**: 文档的原始Json数据
- **_version**: 文档的版本号，修改删除操作_version都会自增1
- **_seq_no**: 和_version一样，一旦数据发生更改，数据也一直是累计的。Shard级别严格递增，保证后写入的Doc的_seq_no大于先写入的Doc的_seq_no。
- **_primary_term**: _primary_term主要是用来恢复数据时处理当多个文档的_seq_no一样时的冲突，避免Primary Shard上的写入被覆盖。每当Primary Shard发生重新分配时，比如重启，Primary选举等，_primary_term会递增1。

ElasticSearch索引操作

<https://www.elastic.co/guide/en/elasticsearch/reference/7.17/index.html>

创建索引

索引命名必须小写，不能以下划线开头

格式: PUT /索引名称

```
1 #创建索引
2 PUT /es_db
3
4 #创建索引时可以设置分片数和副本数
5 PUT /es_db
6 {
7   "settings" : {
```



```
8     "number_of_shards" : 3,
9     "number_of_replicas" : 2
10  }
11  }
12
13  #修改索引配置
14  PUT /es_db/_settings
15  {
16    "index" : {
17      "number_of_replicas" : 1
18    }
19  }
```

#创建索引

PUT /es_db

```
1 {
2   "acknowledged" : true,
3   "shards_acknowledged" : true,
4   "index" : "es_db"
5 }
6
```

查询索引

格式: GET /索引名称

```
1  #查询索引
2  GET /es_db
3
4  #es_db是否存在
5  HEAD /es_db
```

```
#查询索引
GET /es_db

1 {
2   "es_db" : {
3     "aliases" : { },
4     "mappings" : { },
5     "settings" : {
6       "index" : {
7         "routing" : {
8           "allocation" : {
9             "include" : {
10              "_tier_preference" : "data_content"
11            }
12          }
13        },
14        "number_of_shards" : "1",
15        "provided_name" : "es_db",
16        "creation_date" : "1652445122257",
17        "number_of_replicas" : "1",
18        "uuid" : "y7Km2u8ATqyoIFYxkNS60A",
19        "version" : {
20          "created" : "7170399"
21        }
22      }
23    }
24  }
25 }
```

删除索引

格式: DELETE /索引名称

```
1 DELETE /es_db
```

ElasticSearch文档操作

示例数据

```
1 PUT /es_db
2 {
3   "settings" : {
4     "index" : {
5       "analysis.analyzer.default.type": "ik_max_word"
6     }
7   }
8 }
9
10 PUT /es_db/_doc/1
11 {
12   "name": "张三",
13   "sex": 1,
14   "age": 25,
15   "address": "广州天河公园",
16   "remark": "java developer"
17 }
18 PUT /es_db/_doc/2
```


```
19 {
20   "name": "李四",
21   "sex": 1,
22   "age": 28,
23   "address": "广州荔湾大厦",
24   "remark": "java assistant"
25 }
26
27 PUT /es_db/_doc/3
28 {
29   "name": "王五",
30   "sex": 0,
31   "age": 26,
32   "address": "广州白云山公园",
33   "remark": "php developer"
34 }
35
36 PUT /es_db/_doc/4
37 {
38   "name": "赵六",
39   "sex": 0,
40   "age": 22,
41   "address": "长沙橘子洲",
42   "remark": "python assistant"
43 }
44
45 PUT /es_db/_doc/5
46 {
47   "name": "张龙",
48   "sex": 0,
49   "age": 19,
50   "address": "长沙麓谷企业广场",
51   "remark": "java architect assistant"
52 }
53
54 PUT /es_db/_doc/6
55 {
56   "name": "赵虎",
57   "sex": 1,
58   "age": 32,
```

```
59 "address": "长沙麓谷兴工国际产业园",
60 "remark": "java architect"
61 }
```

添加 (索引) 文档

- 格式: [PUT | POST] /索引名称/[_doc | _create]/id

```
1 # 创建文档,指定id
2 # 如果id不存在,创建新的文档,否则先删除现有文档,再创建新的文档,版本会增加
3 PUT /es_db/_doc/1
4 {
5   "name": "张三",
6   "sex": 1,
7   "age": 25,
8   "address": "广州天河公园",
9   "remark": "java developer"
10 }
11
12 #创建文档,ES生成id
13 POST /es_db/_doc
14 {
15   "name": "张三",
16   "sex": 1,
17   "age": 25,
18   "address": "广州天河公园",
19   "remark": "java developer"
20 }
```



The screenshot shows a REST client interface. On the left, a PUT request is defined with the URL `/es_db/_doc/1` and a JSON body: `{ "name": "张三", "sex": 1, "age": 25, "address": "广州天河公园", "remark": "java developer" }`. On the right, the response is displayed as a JSON object: `{ "_index": "es_db", "_type": "_doc", "_id": "1", "_version": 1, "result": "created", "_shards": { "_total": 2, "successful": 1, "failed": 0 }, "_seq_no": 0, "_primary_term": 1 }`.

注意:POST和PUT都能起到创建/更新的作用, PUT需要对一个具体的资源进行操作也就是要确定id才能进行更新/创建, 而POST是可以针对整个资源集合进行操作的, 如果不写id就

由ES生成一个唯一id进行创建新文档，如果填了id那就针对这个id的文档进行创建/更新

```
#创建文档，ES生成id
POST /es_db/_doc
{
  "name": "张三",
  "sex": 1,
  "age": 25,
  "address": "广州天河公园",
  "remark": "java developer"
}
```

不指定id,ES自动生成id

```
1 {
2   "_index" : "es_db",
3   "_type" : "_doc",
4   "_id" : "W3e5vYABfihbfnWAX16H",
5   "_version" : 1,
6   "result" : "created",
7   "shards" : {
8     "total" : 2,
9     "successful" : 1,
10    "failed" : 0
11  },
12   "_seq_no" : 37,
13   "_primary_term" : 5
14 }
15
```

Create -如果ID已经存在，会失败

```
#创建文档
PUT /es_db/_create/1
{
  "name": "张三",
  "sex": 1,
  "age": 25,
  "address": "广州天河公园",
  "remark": "java developer"
}
```

id已经存在

```
1 {
2   "error" : {
3     "root_cause" : [
4       {
5         "type" : "version_conflict_engine_exception",
6         "reason" : "[1]: version conflict, document already exists (current version [21])",
7         "index_uuid" : "9qwIVqZ0TYC0jC5ormR8XA",
8         "shard" : "0",
9         "index" : "es_db"
10      }
11     ],
12     "type" : "version_conflict_engine_exception",
13     "reason" : "[1]: version conflict, document already exists (current version [21])",
14     "index_uuid" : "9qwIVqZ0TYC0jC5ormR8XA",
15     "shard" : "0",
16     "index" : "es_db"
17   },
18   "status" : 409
19 }
20
```

修改文档

- 全量更新，整个json都会替换，格式: [PUT | POST] /索引名称/_doc/id

如果文档存在，现有文档会被删除，新的文档会被索引

```
1 # 全量更新，替换整个json
2 PUT /es_db/_doc/1/
3 {
4   "name": "张三",
5   "sex": 1,
6   "age": 25
7 }
8
9 #查询文档
10 GET /es_db/_doc/1
```

```
# PUT全量更新
PUT /es_db/_doc/1
{
  "name": "张三",
  "sex": 1,
  "age": 30
}
```

#查询文档

```
GET /es_db/_doc/1
```

```
1 # PUT /es_db/_doc/1
2 {
3   "_index" : "es_db",
4   "_type" : "_doc",
5   "_id" : "1",
6   "_version" : 6,
7   "result" : "updated",
8   "_shards" : {
9     "total" : 2,
10    "successful" : 1,
11    "failed" : 0
12  },
13   "_seq_no" : 10,
14   "_primary_term" : 1
15 }
16
17 # GET /es_db/_doc/1
18 {
19   "_index" : "es_db",
20   "_type" : "_doc",
21   "_id" : "1",
22   "_version" : 6,
23   "_seq_no" : 10,
24   "_primary_term" : 1,
25   "found" : true,
26   "source" : {
27     "name" : "张三",
28     "sex" : 1,
29     "age" : 30
30   }
31 }
```

- 使用_update部分更新，格式: POST /索引名称/_update/id

update不会删除原来的文档，而是实现真正的数据更新

```
1 # 部分更新：在原有文档上更新
2 # Update -文档必须已经存在，更新只会对相应字段做增量修改
3 POST /es_db/_update/1
4 {
5   "doc": {
6     "age": 28
7   }
8 }
9
10 #查询文档
11 GET /es_db/_doc/1
```

```
# 部分更新
POST /es_db/_update/1
{
  "doc": {
    "age": 28
  }
}

#查询文档
GET /es_db/_doc/1
```

```
2 {
3   "_index" : "es_db",
4   "_type" : "_doc",
5   "_id" : "1",
6   "_version" : 26,
7   "result" : "noop",
8   "_shards" : {
9     "total" : 0,
10    "successful" : 0,
11    "failed" : 0
12  },
13   "_seq_no" : 34,
14   "_primary_term" : 5
15 }

16
17 # GET /es_db/_doc/1
18 {
19   "_index" : "es_db",
20   "_type" : "_doc",
21   "_id" : "1",
22   "_version" : 26,
23   "_seq_no" : 34,
24   "_primary_term" : 5,
25   "found" : true,
26   "_source" : {
27     "name" : "张三",
28     "sex" : 1,
29     "age" : 28,
30     "address" : "广州天河公园",
31     "remark" : "java developer"
32   }
33 }
```

- 使用 `_update_by_query` 更新文档

```
1 POST /es_db/_update_by_query
2 {
3   "query": {
4     "match": {
5       "_id": 1
6     }
7   },
8   "script": {
9     "source": "ctx._source.age = 30"
10  }
11 }
```

```
#更新符合条件的文档
POST /es_db/_update_by_query
{
  "query": {
    "match": {
      "_id": 1
    }
  },
  "script": {
    "source": "ctx._source.age = 30"
  }
}
```

```
#查询文档
GET /es_db/_doc/1
```

#并发修改，乐观锁实现

```
2 {
3   "took" : 225,
4   "timed_out" : false,
5   "total" : 1,
6   "updated" : 1,
7   "deleted" : 0,
8   "batches" : 1,
9   "version_conflicts" : 0,
10  "noops" : 0,
11  "retries" : {
12    "bulk" : 0,
13    "search" : 0
14  },
15  "throttled_millis" : 0,
16  "requests_per_second" : -1.0,
17  "throttled_until_millis" : 0,
18  "failures" : [ ]
19 }
20
21 # GET /es_db/_doc/1
22 {
23   "_index" : "es_db",
24   "_type" : "doc",
25   "_id" : "1",
26   "_version" : 32,
27   "_seq_no" : 43,
28   "_primary_term" : 5,
29   "found" : true,
30   "_source" : {
31     "sex" : 1,
32     "name" : "张三",
33     "age" : 30
34 }
```

并发场景下修改文档

_seq_no和_primary_term是对_version的优化，7.X版本的ES默认使用这种方式控制版本，所以当在高并发环境下使用乐观锁机制修改文档时，要带上当前文档的_seq_no和_primary_term进行更新：

```
1 POST /es_db/_doc/2?if_seq_no=21&if_primary_term=6
2 {
3   "name": "李四xxx"
4 }
```

如果版本号不对，会抛出版本冲突异常，如下图：

```
POST /es_db/_doc/2?if_seq_no=21&if_primary_term=6
{
  "name": "李四xxx"
}

1 {
2   "error": {
3     "root_cause": [
4       {
5         "type": "version_conflict_engine_exception",
6         "reason": "[2]: version conflict, required seqNo [21], primary term [6]. current document has seqNo [22] and primary term [6]",
7         "index_uuid": "Y6xCnX2XRGa12Qg6zMBKYg",
8         "shard": "0",
9         "index": "es_db"
10      }
11     ],
12     "type": "version_conflict_engine_exception",
13     "reason": "[2]: version conflict, required seqNo [21], primary term [6]. current document has seqNo [22] and primary term [6]",
14     "index_uuid": "Y6xCnX2XRGa12Qg6zMBKYg",
15     "shard": "0",
16     "index": "es_db"
17   },
18   "status": 409
19 }
20
```

查询文档

- 根据id查询文档，格式: GET /索引名称/_doc/id

```
1 GET /es_db/_doc/1
```


- 条件查询 `_search`, 格式: `/索引名称/_doc/_search`

```
1 # 查询前10条文档
2 GET /es_db/_doc/_search
3
```

ES Search API提供了两种条件查询搜索方式:

- REST风格的请求URI, 直接将参数带过去
- 封装到request body中, 这种方式可以定义更加易读的JSON格式

```
1 #通过URI搜索, 使用“q”指定查询字符串, “query string syntax” KV键值对
2
3 #条件查询, 如要查询age等于28岁的 _search?q=*:***
4 GET /es_db/_doc/_search?q=age:28
5
6 #范围查询, 如要查询age在25至26岁之间的 _search?q=***[** TO **] 注意: TO 必须为大写
7 GET /es_db/_doc/_search?q=age[25 TO 26]
8
9 #查询年龄小于等于28岁的 :<=
10 GET /es_db/_doc/_search?q=age:<=28
11 #查询年龄大于28前的 :>
12 GET /es_db/_doc/_search?q=age:>28
13
14 #分页查询 from=*&size=*
15 GET /es_db/_doc/_search?q=age[25 TO 26]&from=0&size=1
16
17 #对查询结果只输出某些字段 _source=字段, 字段
18 GET /es_db/_doc/_search?_source=name,age
19
20 #对查询结果排序 sort=字段:desc/asc
21 GET /es_db/_doc/_search?sort=age:desc
```

通过请求体的搜索方式会在后面课程详细讲解 (DSL)

```
1 GET /es_db/_search
2 {
3   "query": {
4     "match": {
5       "address": "广州白云"
6     }
7   }
8 }
```

删除文档

格式: DELETE /索引名称/_doc/id

```
1 DELETE /es_db/_doc/1
```

ElasticSearch文档批量操作

批量操作可以减少网络连接所产生的开销，提升性能

- 支持在一次API调用中，对不同的索引进行操作
- 可以在URI中指定Index，也可以在请求的Payload中进行
- 操作中单条操作失败，并不会影响其他操作
- 返回结果包括了每一条操作执行的结果

批量写入

批量对文档进行写操作是通过_bulk的API来实现的

- 请求方式：POST
- 请求地址：_bulk
- 请求参数：通过_bulk操作文档，一般至少有两行参数(或偶数行参数)
 - 第一行参数为指定操作的类型及操作的对象(index,type和id)
 - 第二行参数才是操作的数据

参数类似于：

```
1 {"actionName":{"_index":"indexName", "_type":"typeName", "_id":"id"}}
2 {"field1":"value1", "field2":"value2"}
```

- actionName：表示操作类型，主要有create, index, delete和update

批量创建文档create

```
1 POST _bulk
2 {"create":{"_index":"article", "_type":"_doc", "_id":3}}
3 {"id":3,"title":"fox老师","content":"fox老师666","tags":["java", "面向对象"], "create_time":1554015482530}
4 {"create":{"_index":"article", "_type":"_doc", "_id":4}}
5 {"id":4,"title":"mark老师","content":"mark老师NB","tags":["java", "面向对象"], "create_time":1554015482530}
```

普通创建或全量替换index

```
1 POST _bulk
2 {"index":{"_index":"article", "_type":"_doc", "_id":3}}
```

```

3 {"id":3,"title":"图灵徐庶老师","content":"图灵学院徐庶老师666","tags":["java", "面向对象"],"create_time":1554015482530}
4 {"index":{"_index":"article", "_type":"_doc", "_id":4}}
5 {"id":4,"title":"图灵诸葛老师","content":"图灵学院诸葛老师NB","tags":["java", "面向对象"],"create_time":1554015482530}

```

- 如果原文档不存在，则是创建
- 如果原文档存在，则是替换(全量修改原文档)

批量删除delete

```

1 POST _bulk
2 {"delete":{"_index":"article", "_type":"_doc", "_id":3}}
3 {"delete":{"_index":"article", "_type":"_doc", "_id":4}}

```

批量修改update

```

1 POST _bulk
2 {"update":{"_index":"article", "_type":"_doc", "_id":3}}
3 {"doc":{"title":"ES大法必修内功"}}
4 {"update":{"_index":"article", "_type":"_doc", "_id":4}}
5 {"doc":{"create_time":1554018421008}}

```

组合应用

```

1 POST _bulk
2 {"create":{"_index":"article", "_type":"_doc", "_id":3}}
3 {"id":3,"title":"fox老师","content":"fox老师666","tags":["java", "面向对象"],"create_time":1554015482530}
4 {"delete":{"_index":"article", "_type":"_doc", "_id":3}}
5 {"update":{"_index":"article", "_type":"_doc", "_id":4}}
6 {"doc":{"create_time":1554018421008}}

```

批量读取

es的批量查询可以使用mget和msearch两种。其中mget是需要我们知道它的id，可以指定不同的index，也可以指定返回值source。msearch可以通过字段查询来进行一个批量的查找。

_mget

```

1 #可以通过ID批量获取不同index和type的数据
2 GET _mget
3 {
4   "docs": [
5     {
6       "_index": "es_db",

```

```
7  "_id": 1
8  },
9  {
10  "_index": "article",
11  "_id": 4
12  }
13  ]
14  }
15
16  #可以通过ID批量获取es_db的数据
17  GET /es_db/_mget
18  {
19  "docs": [
20  {
21  "_id": 1
22  },
23  {
24  "_id": 4
25  }
26  ]
27  }
28  #简化后
29  GET /es_db/_mget
30  {
31  "ids":["1","2"]
32  }
```

```
#可以通过ID批量获取不同index和type的数据
GET _mget
{
  "docs": [
    {
      "_index": "es_db",
      "_type": "_doc",
      "_id": 1
    },
    {
      "_index": "article",
      "_type": "_doc",
      "_id": 4
    }
  ]
}
```

```
2 {
3   "docs" : [
4     {
5       "_index" : "es_db",
6       "_type" : "_doc",
7       "_id" : "1",
8       "_version" : 32,
9       "_seq_no" : 43,
10      "_primary_term" : 5,
11      "found" : true,
12      "_source" : {
13        "sex" : 1,
14        "name" : "张三",
15        "age" : 30
16      }
17    },
18    {
19      "_index" : "article",
20      "_type" : "_doc",
21      "_id" : "4",
22      "_version" : 3,
23      "_seq_no" : 5,
24      "_primary_term" : 1,
25      "found" : true,
26      "_source" : {
27        "id" : 4,
28        "title" : "图灵诸葛老师",
29        "content" : "图灵学院诸葛老师NB",
30        "tags" : [
31          "java",
32          "面向对象"
33        ],
34        "create_time" : 1554018421008
35      }
36    }
37  ]
38 }
```

_msearch

在_msearch中，请求格式和bulk类似。查询一条数据需要两个对象，第一个设置index和type，第二个设置查询语句。查询语句和search相同。如果只是查询一个index，我们可以在url中带上index，这样，如果查该index可以直接用空对象表示。

```
1 GET /es_db/_msearch
2 {}
3 {"query" : {"match_all" : {}}, "from" : 0, "size" : 2}
4 {"index" : "article"}
5 {"query" : {"match_all" : {}}}
```

```
#批量读取
GET /es_db/_msearch
{}
{"query" : {"match_all" : {}}, "from" : 0, "size" : 2}
{"index" : "article"}
{"query" : {"match_all" : {}}}
```

```
1 {
2   "took" : 1,
3   "responses" : [
4     {
5       "took" : 1,
6       "timed_out" : false,
7       "_shards" : {
8         "total" : 1,
9         "successful" : 1,
10        "skipped" : 0,
11        "failed" : 0
12      },
13      "hits" : {
14        "total" : {
15          "value" : 1,
16          "relation" : "eq"
17        },
18        "max_score" : 1.0,
19        "hits" : [ ]
20      },
21      "status" : 200
22    }
23  ]
24 }
```

