ElasticSearch7入门与进阶实战

主讲老师：图灵课堂Fox老师

ES版本： v7.17.3

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

```
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

图灵课堂Fox

CSDN 博客 下载·课程 学习 问答 社区 插件 认证 开源

elasticsearch快速入门 C一下

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详解，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硬气，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

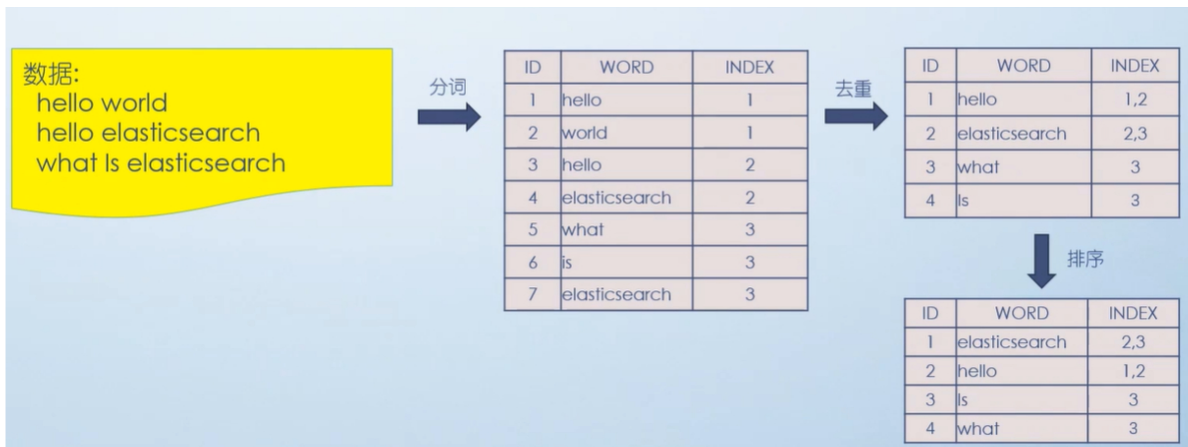
搜索原理简单概括的话可以分为这么几步：

- 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。

图灵课堂Fox



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。

关键词	文章ID
JAVA	1，2
设计模式	1,2
多线程	2

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

图灵课堂FOX

搜索引擎排名：

☐ include secondary database models 26 systems in ranking, May 2022

Rank May 2022	Rank Apr 2022	Rank May 2021	DBMS	Database Model	Score May 2022	Score Apr 2022	Score 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。

图灵课堂FOX

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 - Elasticseach 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

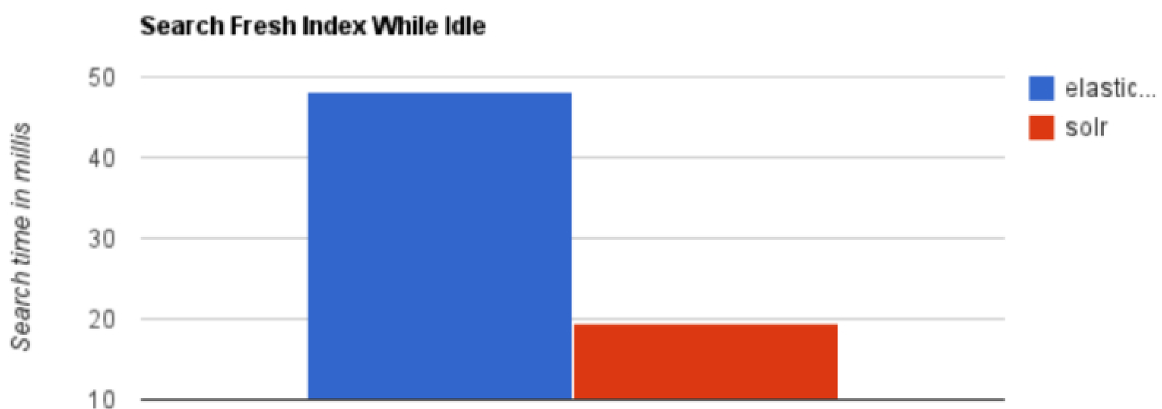
8.x新特性

- 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

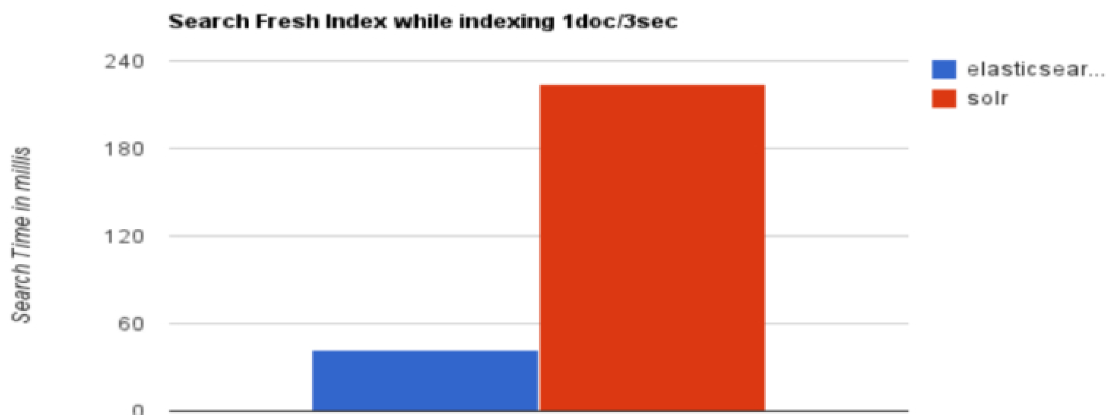
Solr 是第一个基于 Lucene 核心库功能完备的搜索引擎产品，诞生远早于 Elasticsearch。

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。

图灵课堂Fox



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。

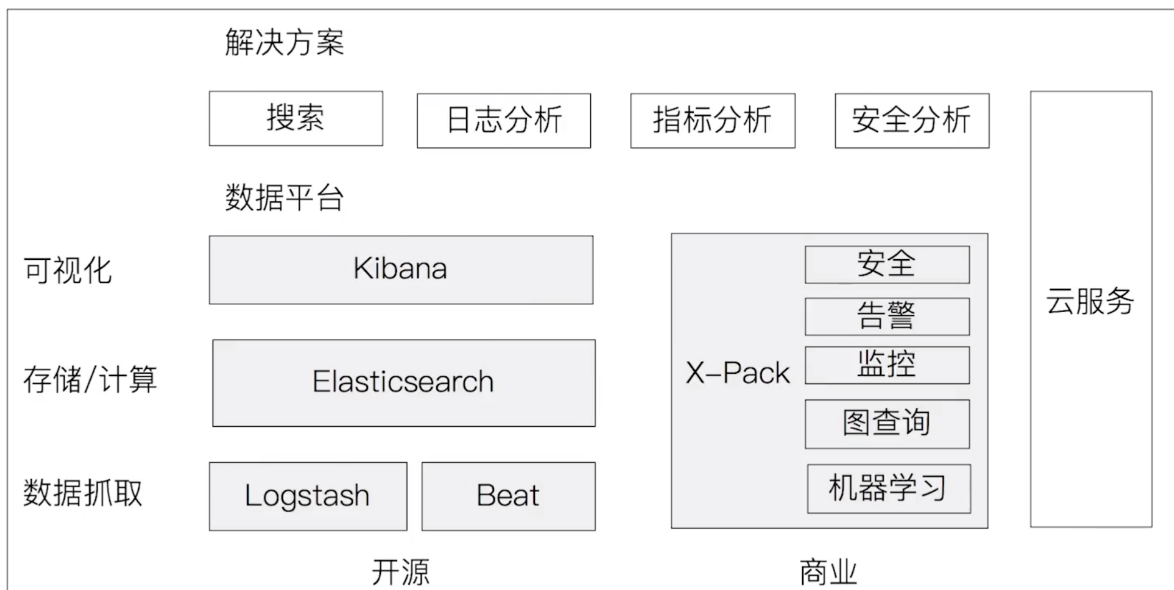
总结：

- 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。

图灵课堂FOX



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

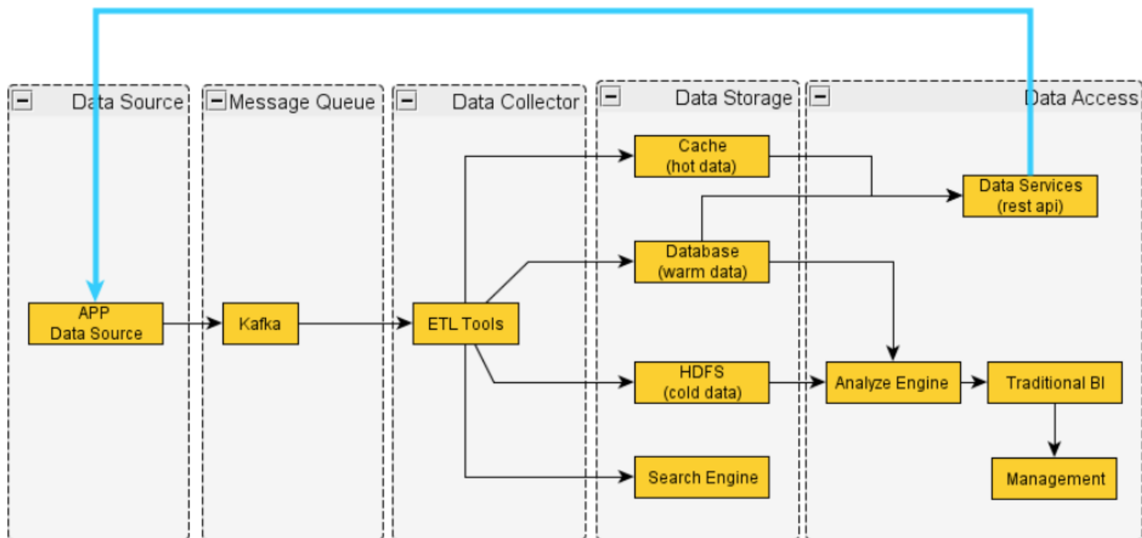
图灵课堂FOX

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

图灵课堂FOX

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

图灵课堂FOX

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，说白一点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。

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

修改JVM配置

修改config/jvm.option配置文件，调整jvm堆内存大小

```
vim jvm.options
-Xms4g
-Xmx4g
```

配置的建议

- Xms和Xms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

图灵课堂FOX

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

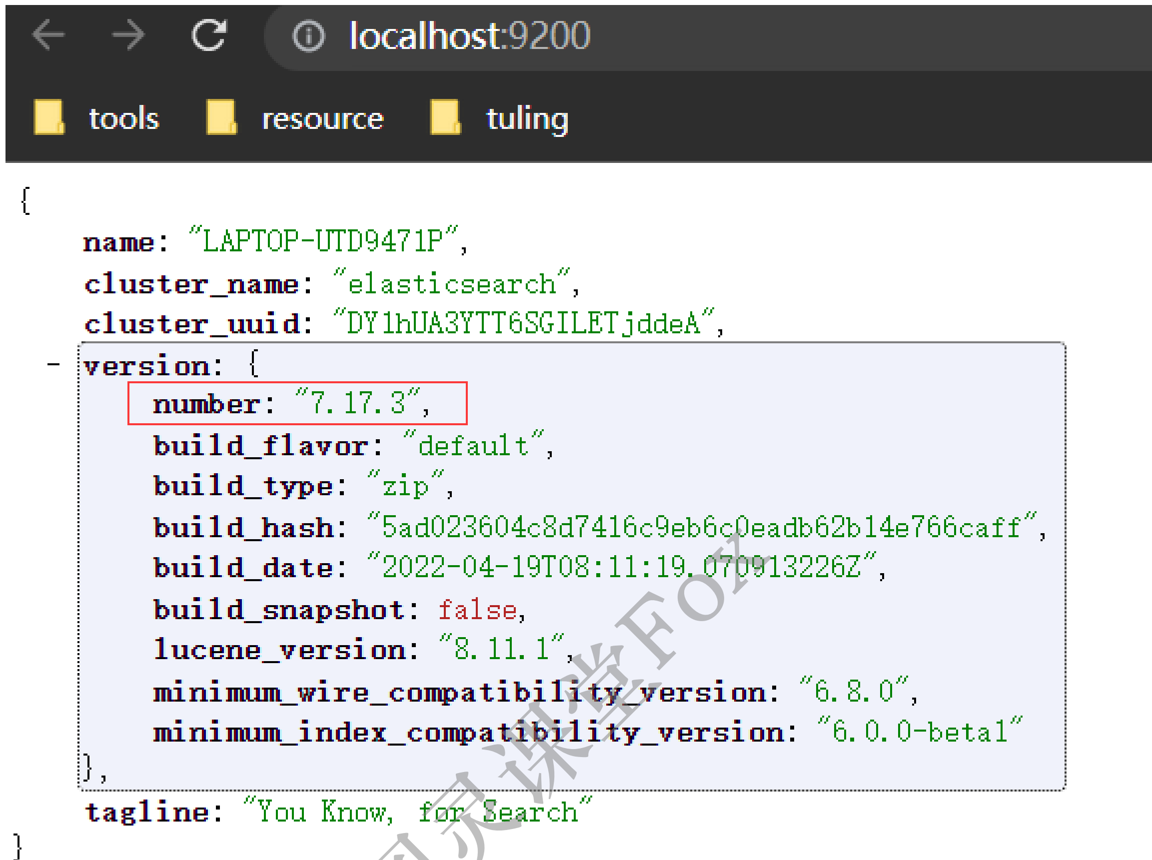
```
#非root用户
bin/elasticsearch
# -d 后台启动
bin/elasticsearch -d
```

```
[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.j
ar:7.17.3]
```

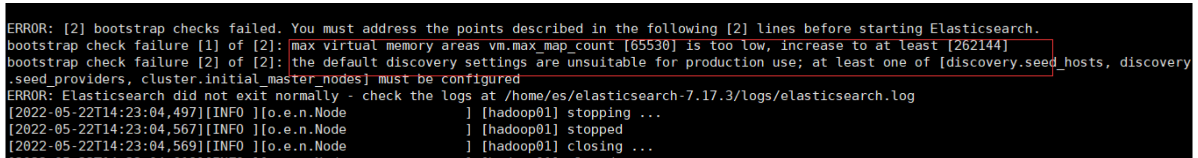
注意：es默认不能用root用户启动，生产环境建议为elasticsearch创建用户。

```
#为elaticsearch创建用户并赋予相应权限
adduser es
passwd es
chown -R es:es .
elasticsearch-17.3
```

运行http://localhost:9200/



如果ES服务启动异常，会有提示：



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

```
\#切换到root用户
vim /etc/security/limits.conf

末尾添加如下配置：
*     soft  nofile  65536
*     hard  nofile  65536
*     soft  nproc   4096
*     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

```
vim /etc/security/limits.d/20-nproc.conf
改为如下配置:
* 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

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

```
vim config/elasticsearch.yml
#添加配置
discovery.seed_hosts: ["127.0.0.1"]
cluster.initial_master_nodes: ["node-1"]
```

图灵课堂Fox

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。

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
April 21, 2022
See Release Notes
Download

修改Kibana.yml

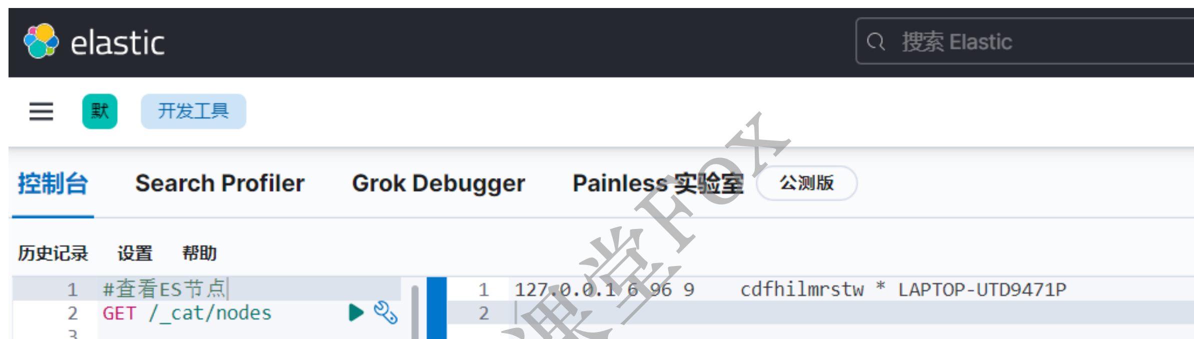
```
vim config/kibana.yml
server.port: 5601
#服务器ip
server.host: "localhost"
#elasticsearch的访问地址
elasticsearch.hosts: ["http://localhost:9200"]
#Kibana汉化
i18n.locale: "zh-CN"
```

运行Kibana

bin/kibana #后台启动

```
nohup  bin/kibana &
```

访问Kibana: http://localhost:5601/



图灵课堂Fox

cat API

```
/_cat/allocation          #查看单节点的shard分配整体情况
/_cat/shards              #查看各shard的详细情况
/_cat/shards/{index}      #查看指定分片的详细情况
/_cat/master              #查看master节点信息
/_cat/nodes               #查看所有节点信息
/_cat/indices             #查看集群中所有index的详细信息
/_cat/indices/{index}       #查看集群中指定index的详细信息
/_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
/_cat/segments/{index}#查看指定index的segment详细信息
/_cat/count               #查看当前集群的doc数量
/_cat/count/{index}       #查看指定索引的doc数量
/_cat/recovery            #查看集群内每个shard的recovery过程.调整replica。
/_cat/recovery/{index}#查看指定索引shard的recovery过程
/_cat/health              #查看集群当前状态: 红、黄、绿
/_cat/pending_tasks       #查看当前集群的pending task
/_cat/aliases             #查看集群中所有alias信息,路由配置等
/_cat/aliases/{alias} #查看指定索引的alias信息
/_cat/thread_pool         #查看集群各节点内部不同类型的threadpool的统计信息,
/_cat/plugins             #查看集群各个节点上的plugin信息
/_cat/fielddata           #查看当前集群各个节点的fielddata内存使用情况
/_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
/_cat/nodeattrs                #查看单节点的自定义属性
/_cat/repositories             #输出集群中注册快照存储库
/_cat/templates                #输出当前正在存在的模板信息
```

Elasticsearch安装插件

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

```
#查看已安装插件
bin/elasticsearch-plugin list
#安装插件
bin/elasticsearch-plugin install analysis-icu
#删除插件
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

```
POST _analyze
{
    "analyzer":"icu_analyzer",
    "text":"中华人民共和国"
}
```

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

```
{
  "tokens" : [
    {
      "token" : "中华",
      "start_offset" : 0,
      "end_offset" : 2,
      "type" : "<IDEOGRAPHIC>",
      "position" : 0
    },
    {
      "token" : "人民",
      "start_offset" : 2,
      "end_offset" : 4,
      "type" : "<IDEOGRAPHIC>",
      "position" : 1
    },
    {
      "token" : "共和国",
      "start_offset" : 4,
      "end_offset" : 7,
      "type" : "<IDEOGRAPHIC>",
      "position" : 2
    }
  ]
}
```

图灵课堂Fox

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

```
unzip elasticsearch-analysis-ik-7.17.3.zip -d analysis-ik
rm elasticsearch-analysis-ik-7.17.3.zip
```

测试分词效果

```
#ES的默认分词设置是standard，会单字拆分
```

```
POST _analyze
{
    "analyzer":"standard",
    "text":"中华人民共和国"
}

#ik_smart:会做最粗粒度的拆
POST _analyze
{
    "analyzer": "ik_smart",
    "text": "中华人民共和国"
 }

#ik_max_word:会将文本做最细粒度的拆分
POST _analyze
{
    "analyzer":"ik_max_word",
    "text":"中华人民共和国"
}
```

创建索引时可以指定IK分词器作为默认分词器

```
PUT /es_db
{
    "settings" : {
        "index" : {
            "analysis.analyzer.default.type": "ik_max_word"
        }
    }
}
```

图灵课堂Fox

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。

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

图灵课堂Fox

```
GET /es_db/_doc/2
```

```
{
  "_index" : "es_db",
  "_type" : "_doc",
  "_id" : "2",
  "_version" : 2,
  "_seq_no" : 15,
  "_primary_term" : 6,
  "found" : true,
  "_source" : {
    "name" : "李四",
    "sex" : 1,
    "age" : 28,
    "address" : "广州荔湾大厦",
    "remark" : "java assistant"
  }
}
```

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。

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：

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

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"
}
```

```
{
  "error" : {
    "root_cause" : [
      {
        "type" : "version_conflict_engine_exception",
        "reason" : "[2]: version conflict, required seqNo [21], primary term [6]. current document has seqNo [22] and primary term [6]",
        "index_uuid" : "Y6xCnX2XRGal2Qg6zWBKYg",
        "shard" : "0",
        "index" : "es_db"
      }
    ],
    "type" : "version_conflict_engine_exception",
    "reason" : "[2]: version conflict, required seqNo [21], primary term [6]. current document has seqNo [22] and primary term [6]",
    "index_uuid" : "Y6xCnX2XRGal2Qg6zWBKYg",
    "shard" : "0",
    "index" : "es_db"
  },
  "status" : 409
}
```

ElasticSearch索引操作

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

图灵课堂Fox

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

```
#创建索引
PUT /es_db

#创建索引时可以设置分片数和副本数
PUT /es_db
{
    "settings" : {
        "number_of_shards" : 3,
        "number_of_replicas" : 2
    }
}

#修改索引配置
PUT /es_db/_settings
{
    "index" : {
        "number_of_replicas" : 1
    }
}
```

```
#创建索引
PUT /es_db
```

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

图灵课堂Fox

查询索引

格式: GET /索引名称

```
#查询索引
GET /es_db

#es_db是否存在
HEAD /es_db
```

```
#查询索引
GET /es_db
```

```
{
  "es_db" : {
    "aliases" : { },
    "mappings" : { },
    "settings" : {
      "index" : {
        "routing" : {
          "allocation" : {
            "include" : {
              "_tier_preference" : "data_content"
            }
          }
        },
        "number_of_shards" : "1",
        "provided_name" : "es_db",
        "creation_date" : "1652445122257",
        "number_of_replicas" : "1",
        "uuid" : "y7Km2u8ATqyoIFYxkNS6OA",
        "version" : {
          "created" : "7170399"
        }
      }
    }
  }
}
```

删除索引

格式: DELETE /索引名称

```
DELETE /es_db
```

图灵课堂FOX

ElasticSearch文档操作

示例数据

```
PUT /es_db
{
    "settings" : {
        "index" : {
            "analysis.analyzer.default.type": "ik_max_word"
        }
    }
}

PUT /es_db/_doc/1
{
"name": "张三",
"sex": 1,
"age": 25,
"address": "广州天河公园",
"remark": "java developer"
}
PUT /es_db/_doc/2
{
"name": "李四",
"sex": 1,
"age": 28,
"address": "广州荔湾大厦",
"remark": "java assistant"
}
```

```
PUT /es_db/_doc/3
{
"name": "王五",
"sex": 0,
"age": 26,
"address": "广州白云山公园",
"remark": "php developer"
}

PUT /es_db/_doc/4
{
"name": "赵六",
"sex": 0,
"age": 22,
"address": "长沙橘子洲",
"remark": "python assistant"
}

PUT /es_db/_doc/5
{
"name": "张龙",
"sex": 0,
"age": 19,
"address": "长沙麓谷企业广场",
"remark": "java architect assistant"
}

PUT /es_db/_doc/6
{
"name": "赵虎",
"sex": 1,
"age": 32,
"address": "长沙麓谷兴工国际产业园",
"remark": "java architect"
}
```

图灵课堂FOX

添加文档

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

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

#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"
}
```

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

```
{
  "_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

```
{
  "_index" : "es_db",
  "_type" : "_doc",
  "_id" : "W3e5vYABfihbfNWAXl6H",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 37,
  "_primary_term" : 5
}
```

图灵课堂Fox

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

```
{
  "error" : {
    "root_cause" : [
      {
        "type" : "version_conflict_engine_exception",
        "reason" : "[1]: version conflict, document already exists (current version [21])",
        "index_uuid" : "9qwIvqZ0TYCOjC5ormR8XA",
        "shard" : "0",
        "index" : "es_db"
      }
    ],
    "type" : "version_conflict_engine_exception",
    "reason" : "[1]: version conflict, document already exists (current version [21])",
    "index_uuid" : "9qwIvqZ0TYCOjC5ormR8XA",
    "shard" : "0",
    "index" : "es_db"
  },
  "status" : 409
}
```

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

```
# 全量更新，替换整个json
PUT /es_db/_doc/1/
{
"name": "张三",
"sex": 1,
"age": 25
}

#查询文档
GET /es_db/_doc/1
```

- 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

```
# 部分更新：在原有文档上更新
# Update -文档必须已经存在，更新只会对相应字段做增量修改
POST /es_db/_update/1
{
  "doc": {
    "age": 28
  }
}

#查询文档
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

```
POST /es_db/_update_by_query
{
  "query": {
    "match": {
      "_id": 1
    }
  },
  "script": {
    "source": "ctx._source.age = 30"
  }
}
```

图灵课堂Fox

```
#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   }
```

图灵课堂FOX

查询文档

- 根据id查询文档，格式: GET /索引名称/_doc/id

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

- 条件查询 *search*，格式：/索引名称/doc/_search

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

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

```
#通过URI搜索，使用“q”指定查询字符串，“query string syntax” KV键值对

#条件查询，如要查询age等于28岁的 _search?q=*:***
GET /es_db/_doc/_search?q=age:28

#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
GET /es_db/_doc/_search?q=age[25 TO 26]

#查询年龄小于等于28岁的 :<=
GET /es_db/_doc/_search?q=age:<=28
#查询年龄大于28前的 :>
```

```
GET /es_db/_doc/_search?q=age:>28

#分页查询 from=*&size=*
GET /es_db/_doc/_search?q=age[25 TO 26]&from=0&size=1

#对查询结果只输出某些字段 _source=字段,字段
GET /es_db/_doc/_search?_source=name,age

#对查询结果排序 sort=字段:desc/asc
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

删除文档

格式: DELETE /索引名称/_doc/id

```
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
- 可以再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：

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

图灵课堂FOX

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

已存在的文档会产生冲突

```
"took" : 0,
"errors" : true,
"items" : [
  {
    "create" : {
      "_index" : "article",
      "_type" : "_doc",
      "_id" : "3",
      "status" : 409,
      "error" : {
        "type" : "version_conflict_engine_exception",
        "reason" : "[3]: version conflict, document already exists (current version [1])",
        "index_uuid" : "JBtRrVLQRbKolj_emiJqgA",
        "shard" : "0",
        "index" : "article"
      }
    }
```

普通创建或全量替换index

```
POST _bulk
{"index":{"_index":"article", "_type":"_doc", "_id":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}
{"index":{"_index":"article", "_type":"_doc", "_id":4}}
{"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

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

批量修改update

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

图灵课堂Fox

组合应用

```
POST _bulk
{"delete":{"_index":"article", "_type":"_doc", "_id":3}}
{"create":{"_index":"article", "_type":"_doc", "_id":3}}
{"title":"fox老师","content":"fox老师666","tags":["java", "面向对
象"],"create_time":1554015482530}
{"update":{"_index":"article", "_type":"_doc", "_id":4}}
{"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

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

#可以通过ID批量获取es_db的数据
GET /es_db/_mget
{
"docs": [
{
"_id": 1
},
{
"_id": 4
}
]
}
#简化后
GET /es_db/_mget
{
 "ids":["1","2"]
 }
```

图灵课堂FOX

```
#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       }
```

_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。

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

图灵课堂Fox

```
#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      {⟺},
50     {
51       "took" : 1,
52       "timed_out" : false,
53       "_shards" : {
54         "total" : 1,
55         "successful" : 1,
56         "skipped" : 0,
57         "failed" : 0
58       },
59       "hits" : {
60         "total" : {
61           "value" : 1,
62           "relation" : "eq"
63         },
64         "max_score" : 1.0,
65         "hits" : [⟺]
83       },
84       "status" : 200
85     }
86   ]
87 }
```

ES检索原理分析

索引的原理

索引是加速数据查询的重要手段，其核心原理是通过不断的缩小想要获取数据的范围来筛选出最终想要的结果，同时把随机的事件变成顺序的事件。

磁盘IO与预读

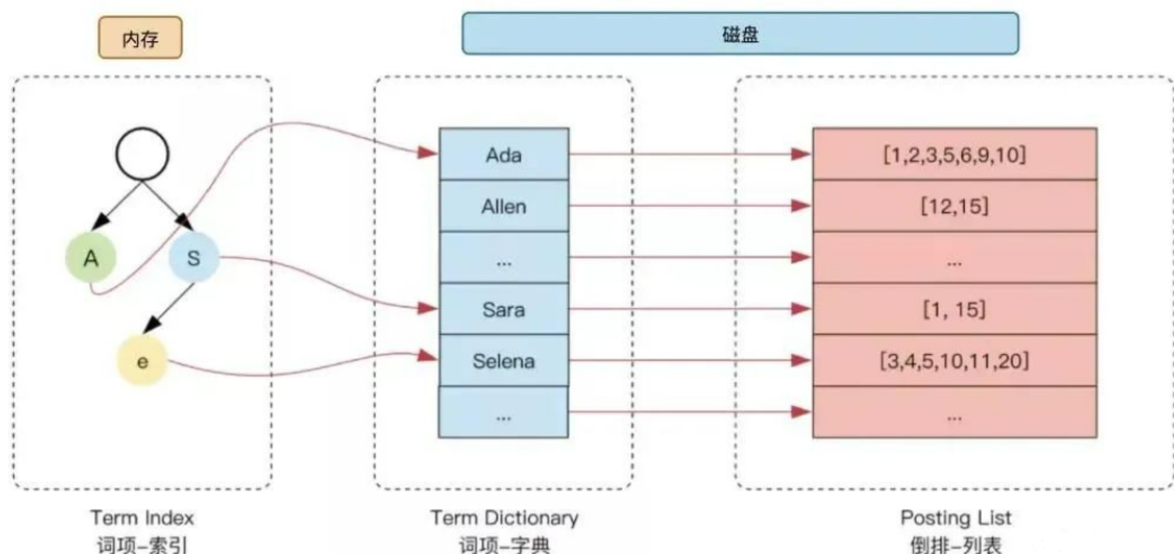
磁盘IO是程序设计中非常高昂的操作，也是影响程序性能的重要因素，因此应当尽量避免过多的磁盘IO，有效的利用内存可以大大的提升程序的性能。在操作系统层面，发生一次IO时，不光把当前磁盘地址的数据，而是把相邻的数据也都读取到内存缓冲区内，局部预读性原理告诉我们，当计算机访问一个地址的数据的时候，与其相邻的数据也会很快被访问到。每一次IO读取的数据我们称之为一页(page)。具体一页有多大数据跟操作系统有关，一般为4k或8k，也就是我们读取一页内的数据时候，实际上才发生了一次IO，这个理论对于索引的数据结构设计非常有帮助。

ES倒排索引

当数据写入 ES 时，数据将会通过 分词 被切分为不同的 term，ES 将 term 与其对应的文档列表建立一种映射关系，这种结构就是 倒排索引。如下图所示：

图灵课堂FOX

为了进一步提升索引的效率，ES 在 term 的基础上利用 term 的前缀或者后缀构建了 term index, 用于对 term 本身进行索引，ES 实际的索引结构如下图所示：



这样当我们去搜索某个关键词时，ES 首先根据它的前缀或者后缀迅速缩小关键词的在 term dictionary 中的范围，大大减少了磁盘IO的次数。2

- 单词词典（Term Dictionary）：记录所有文档的单词，记录单词到倒排列表的关联关系
- 倒排列表(Posting List)-记录了单词对应的文档结合，由倒排索引项组成

- 倒排索引项(Posting)：
-
  - 文档ID
  - 词频TF–该单词在文档中出现的次数，用于相关性评分
  - 位置(Position)-单词在文档中分词的位置。用于短语搜索（match phrase query)
  - 偏移(Offset)-记录单词的开始结束位置，实现高亮显示

```
#ES的默认分词设置是standard，会单字拆分
POST _analyze
{
  "analyzer":"standard",
  "text":"中华人民共和国"
}

#ik_smart:会做最粗粒度的拆
POST _analyze
{
  "analyzer": "ik_smart",
  "text": "中华人民共和国"
 }

#ik_max_word:会将文本做最细粒度的拆分
POST _analyze
{
  "analyzer":"ik_max_word",
  "text":"中华人民共和国"
}
```

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

ES高级查询Query DSL

ES中提供了一种强大的检索数据方式,这种检索方式称之为Query DSL（Domain Specified Language）, Query DSL是利用Rest API传递JSON格式的请求体(RequestBody)数据与ES进行交互，这种方式的丰富查询语法让ES检索变得更强大，更简洁。

图灵课堂Fox

https://www.elastic.co/guide/en/elasticsearch/reference/7.17/query-dsl.html

语法:

```
GET /es_db/_doc/_search {json请求体数据}
可以简化为下面写法
GET /es_db/_search {json请求体数据}
```

示例数据

```
#指定ik分词器
PUT /es_db
{
  "settings" : {
      "index" : {
          "analysis.analyzer.default.type": "ik_max_word"
      }
  }
}

# 创建文档,指定id
PUT /es_db/_doc/1
{
```

```
"name": "张三",
"sex": 1,
"age": 25,
"address": "广州天河公园",
"remark": "java developer"
}
PUT /es_db/_doc/2
{
"name": "李四",
"sex": 1,
"age": 28,
"address": "广州荔湾大厦",
"remark": "java assistant"
}

PUT /es_db/_doc/3
{
"name": "王五",
"sex": 0,
"age": 26,
"address": "广州白云山公园",
"remark": "php developer"
}

PUT /es_db/_doc/4
{
"name": "赵六",
"sex": 0,
"age": 22,
"address": "长沙橘子洲",
"remark": "python assistant"
}

PUT /es_db/_doc/5
{
"name": "张龙",
"sex": 0,
"age": 19,
"address": "长沙麓谷企业广场",
"remark": "java architect assistant"
}

PUT /es_db/_doc/6
{
"name": "赵虎",
"sex": 1,
"age": 32,
"address": "长沙麓谷兴工国际产业园",
"remark": "java architect"
}
```

图灵课堂FOX

查询所有match_all

使用match_all，默认只会返回10条数据。

原因：_search查询默认采用的是分页查询，每页记录数size的默认值为10。如果想显示更多数据，指定size

```
GET /es_db/_search
等同于
GET /es_db/_search
{
"query":{
"match_all":{}
}
}
```

返回指定条数size

size 关键字: 指定查询结果中返回指定条数。 默认返回值10条

```
GET /es_db/_search
{
  "query": {
    "match_all": {}
  },
  "size": 100
}
```

分页查询form

from 关键字: 用来指定起始返回位置，和size关键字连用可实现分页效果

```
GET /es_db/_search
{
  "query": {
    "match_all": {}
  },
  "size": 5,
  "from": 0
}
```

思考： size可以无限增加吗?

测试

```
GET /es_db/_search
{
  "query": {
    "match_all": {}
  },
  "size": 20000
}
```

出现异常：

图灵课堂FOX

```
GET /es_db/_search
{
  "query": {
    "match_all": {}
  },
  "size": 20000
}
```

```
{
  "error" : {
    "root_cause" : [
      {
        "type" : "illegal_argument_exception",
        "reason" : "Result window is too large, from + size must be less than or equal to: [10000] but was [20000]. See the scroll api for a more efficient way to request large data sets. This limit can be set by changing the [index.max_result_window] index level setting."
      }
    ],
    "type" : "search_phase_execution_exception",
    "reason" : "all shards failed",
    "phase" : "query",
    "grouped" : true,
    "failed_shards" : [
      {
        "shard" : 0,
        "index" : "es_db",
        "node" : "6bUwgJ9USSubhnA9gs-_2A",
```

异常原因：

1、查询结果的窗口太大，from + size的结果必须小于或等于10000，而当前查询结果的窗口为20000。

2、可以采用scroll api更高效的请求大量数据集。

3、查询结果的窗口的限制可以通过参数index.max_result_window进行设置。

```
PUT /es_db/_settings
{
  "index.max_result_window" :"20000"
}
#修改现有所有的索引，但新增的索引，还是默认的10000
PUT /_all/_settings
{
  "index.max_result_window" :"20000"
}

#查看所有索引中的index.max_result_window值
GET /_all/_settings/index.max_result_window
```

图灵课堂Fox

注意：参数index.max_result_window主要用来限制单次查询满足查询条件的结果窗口的大小，窗口大小由from + size共同决定。不能简单理解成查询返回给调用方的数据量。这样做主要是为了限制内存的消耗。

比如：from为1000000，size为10，逻辑意义是从满足条件的数据中取1000000到（1000000 + 10）的记录。这时ES一定要先将（1000000 + 10）的记录（即result_window）加载到内存中，再进行分页取值的操作。尽管最后我们只取了10条数据返回给客户端，但ES进程执行查询操作的过程中确需要将（1000000 + 10）的记录都加载到内存中，可想而知对内存的消耗有多大。这也是ES中不推荐采用（from + size）方式进行深度分页的原因。

同理，from为0，size为1000000时，ES进程执行查询操作的过程中确需要将1000000 条记录都加载到内存中再返回给调用方，也会对ES内存造成很大压力。

分页查询Scroll

改动index.max_result_window参数值的大小，只能解决一时的问题，当索引的数据量持续增长时，在查询全量数据时还是会出现问题。而且会增加ES服务器内存大结果集消耗完的风险。最佳实践还是根据异常提示中的采用scroll api更高效的请求大量数据集。

```
#查询命令中新增scroll=1m,说明采用游标查询，保持游标查询窗口一分钟。
#这里由于测试数据量不够，所以size值设置为2。
#实际使用中为了减少游标查询的次数，可以将值适当增大，比如设置为1000。
GET /es_db/_search?scroll=1m
{
    "query": { "match_all": {}},
    "size":  2
}
```

查询结果：

除了返回前2条记录，还返回了一个游标ID值_scroll_id。

采用游标id查询：

```
# scroll_id 的值就是上一个请求中返回的 _scroll_id 的值
GET /_search/scroll
{
    "scroll": "1m",
    "scroll_id" :
"FGluY2x1ZGVfY29udGV4dF91dWlkDXF1ZXJ5QW5kRmV0Y2gBFmNwcVdjblRxUzVhZXlicG9HeU02bWc
AAAAAAABmzRY2YlV3Z0o5VVNTdWJobkE5Z3MtXzJB"
}
```

多次根据scroll_id游标查询，直到没有数据返回则结束查询。采用游标查询索引全量数据，更安全高效，限制了单次对内存的消耗。

指定字段排序sort

注意：会让得分失效

```
GET /es_db/_search
{
  "query": {
    "match_all": {}
  },
  "sort": [
    {
      "age": "desc"
    }
  ]
}

#排序，分页
GET /es_db/_search
{
  "query": {
    "match_all": {}
  },
  "sort": [
    {
      "age": "desc"
```

图灵课堂FOX

```
    }
  ],
  "from": 10,
  "size": 5
}
```

返回指定字段_source

_source 关键字: 是一个数组,在数组中用来指定展示那些字段

```
GET /es_db/_search
{
  "query": {
    "match_all": {}
  },
  "_source": ["name","address"]
}
```

match

match在匹配时会对所查找的关键词进行分词，然后按分词匹配查找

match支持以下参数:

- query : 指定匹配的值
- operator : 匹配条件类型
-
  - and : 条件分词后都要匹配
  - or : 条件分词后有一个匹配即可(默认)
- minmum_should_match : 最低匹配度，即条件在倒排索引中最低的匹配度

```
#模糊匹配 match   分词后or的效果
GET /es_db/_search
{
  "query": {
    "match": {
      "address": "广州白云山公园"
    }
  }
}

# 分词后 and的效果
GET /es_db/_search
{
  "query": {
    "match": {
      "address": {
        "query": "广州白云山公园",
        "operator": "AND"
      }
    }
  }
}
```

图灵课堂FOX

在match中的应用： 当operator参数设置为or时，minnum_should_match参数用来控制匹配的分词的最少数量。

```
# 最少匹配广州，公园两个词
GET /es_db/_search
{
  "query": {
    "match": {
      "address": {
        "query": "广州公园",
        "minimum_should_match": 2
      }
    }
  }
}
```

短语查询match_phrase

match_phrase查询分析文本并根据分析的文本创建一个短语查询。match_phrase 会将检索关键词分词。match_phrase的分词结果必须在被检索字段的分词中都包含，而且顺序必须相同，而且默认必须都是连续的。

```
GET /es_db/_search
{
  "query": {
    "match_phrase": {
      "address": "广州白云山"
    }
  }
}
GET /es_db/_search
{
  "query": {
    "match_phrase": {
      "address": "广州白云"
    }
  }
}
```

图灵课堂FOX

思考：为什么查询广州白云山有数据，广州白云没有数据？

```
    }
  }
 }
}

# match phrase  短语匹配

GET /es_db/_search
{
  "query": {
    "match_phrase": {
      "address": "广州白云山"
    }
  }
}
GET /es_db/_search
{
  "query": {
    "match_phrase": {
      "address": "广州白云"
    }
  }
}

GET /es_db/_search
{
  "query": {
    "match_phrase": {
      "address": {
        "query": "广州云山",
        "slop": 2
      }
```

思考：为什么广州白云山有数据，查广州白云没数据?

```
15    },
16    "max_score" : 4.949977,
17    "hits" : [
18      {
19        "_index" : "es_db",
20        "_type" : "_doc",
21        "_id" : "3",
22        "_score" : 4.949977,
23        "_source" : {
24          "name" : "王五",
25          "sex" : 0,
26          "age" : 26,
27          "address" : "广州白云山公园",
28          "remark" : "php developer"
29        }
30      }
31    ]
32  }
33 }
34
35 # GET /es_db/_search
36 {
37   "took" : 1,
38   "timed_out" : false,
39   "_shards" : {…},
45   "hits" : {
46     "total" : {
47       "value" : 0,
48       "relation" : "eq"
49     },
50     "max_score" : null,
51     "hits" : [ ]
52   }
```

分析原因：

先查看广州白云山公园分词结果，可以知道广州和白云不是相邻的词条，中间会隔一个白云山，而match_phrase匹配的是相邻的词条，所以查询广州白云山有结果，但查询广州白云没有结果。

图灵课堂FOX

```
POST _analyze
{
    "analyzer":"ik_max_word",
    "text":"广州白云山"
}
#结果
{
  "tokens" : [
    {
      "token" : "广州",
      "start_offset" : 0,
      "end_offset" : 2,
      "type" : "CN_WORD",
      "position" : 0
    },
    {
      "token" : "白云山",
      "start_offset" : 2,
      "end_offset" : 5,
      "type" : "CN_WORD",
      "position" : 1
    },
    {
      "token" : "白云",
      "start_offset" : 2,
      "end_offset" : 4,
      "type" : "CN_WORD",
      "position" : 2
    },
    {
      "token" : "云山",
      "start_offset" : 3,
      "end_offset" : 5,
      "type" : "CN_WORD",
```

```
        "position" : 3
      }
    ]
  }
```

如何解决词条间隔的问题？可以借助slop参数，slop参数告诉match_phrase查询词条能够相隔多远时仍然将文档视为匹配。

```
#广州云山分词后相隔为2，可以匹配到结果
GET /es_db/_search
{
  "query": {
    "match_phrase": {
      "address": {
        "query": "广州云山",
        "slop": 2
      }
    }
  }
}
```

多字段查询multi_match

可以根据字段类型，决定是否使用分词查询，得分最高的在前面

图灵课堂Fox

```
GET /es_db/_search
{
  "query": {
    "multi_match": {
      "query": "长沙张龙",
      "fields": [
        "address",
        "name"
      ]
    }
  }
}
```

注意：字段类型分词,将查询条件分词之后进行查询，如果该字段不分词就会将查询条件作为整体进行查询

query_string

允许我们在单个查询字符串中指定AND | OR | NOT条件，同时也和 multi_match query 一样，支持多字段搜索。和match类似，但是match需要指定字段名，query_string是在所有字段中搜索，范围更广泛。

注意: 查询字段分词就将查询条件分词查询，查询字段不分词将查询条件不分词查询

- **未指定字段查询**

```
GET /es_db/_search
{
  "query": {
    "query_string": {
      "query": "张三 OR 橘子洲"
    }
  }
}
```

- **指定单个字段查询**

```
#Query String
GET /es_db/_search
{
  "query": {
    "query_string": {
      "default_field": "address",
      "query": "白云山 OR 橘子洲"
    }
  }
}
```

- **指定多个字段查询**

```
GET /es_db/_search
{
  "query": {
    "query_string": {
      "fields": ["name","address"],
      "query": "张三 OR (广州 AND 王五)"
    }
  }
}
```

图灵课堂FOX

simple_query_string

类似Query String，但是会忽略错误的语法,同时只支持部分查询语法，不支持AND OR NOT，会当作字符串处理。支持部分逻辑：

- + 替代AND
- | 替代OR
- - 替代NOT

```
#simple_query_string 默认的operator是OR
GET /es_db/_search
{
  "query": {
    "simple_query_string": {
      "fields": ["name","address"],
      "query": "广州公园",
      "default_operator": "AND"
    }
  }
```

```
}

GET /es_db/_search
{
  "query": {
    "simple_query_string": {
      "fields": ["name","address"],
      "query": "广州+公园"

    }
  }
}
```

关键词查询Term

Term用来使用关键词查询(精确匹配),还可以用来查询没有被进行分词的数据类型。Term是表达语意的最小单位，搜索和利用统计语言模型进行自然语言处理都需要处理Term。match在匹配时会对所查找的关键词进行分词，然后按分词匹配查找，而term会直接对关键词进行查找。一般模糊查找的时候，多用match，而精确查找时可以使用term。

- ES中默认使用分词器为标准分词器(StandardAnalyzer),标准分词器对于英文单词分词,对于中文单字分词。
- 在ES的Mapping Type 中 keyword , date ,integer, long , double , boolean or ip 这些类型不分词，只有text类型分词。

图灵课堂Fox

```
#关键字查询 term
# 思考： 查询广州白云是否有数据，为什么？
GET /es_db/_search
{
  "query":{
    "term": {
      "address": {
        "value": "广州白云"
      }
    }
  }
}

# 采用term精确查询， 查询字段映射类型为keyword
GET /es_db/_search
{
  "query":{
    "term": {
      "address.keyword": {
        "value": "广州白云山公园"
      }
    }
  }
}
```

在ES中，Term查询，对输入不做分词。会将输入作为一个整体，在倒排索引中查找准确的词项，并且使用相关度算分公式为每个包含该词项的文档进行相关度算分。

```
PUT /product/_bulk
{"index":{"_id":1}}
{"productId":"xxx123","productName":"iPhone"}
{"index":{"_id":2}}
{"productId":"xxx111","productName":"iPad"}

# 思考： 查询iPhone可以查到数据吗？
GET /product/_search
{
  "query":{
    "term": {
      "productName": {
        "value": "iPhone"
      }
    }
  }
}

GET /product/_analyze
{
  "analyzer":"standard",
  "text":"iPhone"
}
```

可以通过 Constant Score 将查询转换成一个 Filtering，避免算分，并利用缓存，提高性能。

- 将Query 转成 Filter，忽略TF-IDF计算，避免相关性算分的开销
- Filter可以有效利用缓存

  ```
  GET /es_db/_search
  {
    "query": {
      "constant_score": {
        "filter": {
          "term": {
            "address.keyword": "广州白云山公园"
          }
        }
      }
    }
  }
  ```

  应用场景：对bool，日期，数字，结构化的文本可以利用term做精确匹配

  ```
  GET /es_db/_search
  {
    "query": {
      "term": {
        "age": {
          "value": 28
        }
      }
    }
  }
  ```

  term处理多值字段，term查询是包含，不是等于

图灵课堂Fox

```
POST /employee/_bulk
{"index":{"_id":1}}
{"name":"小明","interest":["跑步","篮球"]}
{"index":{"_id":2}}
{"name":"小红","interest":["跳舞","画画"]}
{"index":{"_id":3}}
{"name":"小丽","interest":["跳舞","唱歌","跑步"]}

POST /employee/_search
{
  "query": {
    "term": {
      "interest.keyword": {
        "value": "跑步"
      }
    }
  }
}
```

prefix前缀搜索

它会对分词后的term进行前缀搜索。

- 它不会分析要搜索字符串，传入的前缀就是想要查找的前缀
- 默认状态下，前缀查询不做相关度分数计算，它只是将所有匹配的文档返回，然后赋予所有相关分数值为1。它的行为更像是一个过滤器而不是查询。两者实际的区别就是过滤器是可以被缓存的，而前缀查询不行。

prefix的原理：需要遍历所有倒排索引，并比较每个term是否已所指定的前缀开头。

```
GET /es_db/_search
{
  "query": {
    "prefix": {
      "address": {
        "value": "广州白云"
      }
    }
  }
}
```

通配符查询wildcard

通配符查询：工作原理和prefix相同，只不过它不是只比较开头，它能支持更为复杂的匹配模式。

图灵课堂Fox

```
GET /es_db/_search
{
  "query": {
    "wildcard": {
      "address": {
        "value": "*白*"
      }
    }
  }
}
```

范围查询range

- range：范围关键字
- gte 大于等于
- lte 小于等于
- gt 大于
- lt 小于
- now 当前时间

```
POST /es_db/_search
{
  "query": {
    "range": {
      "age": {
        "gte": 25,
        "lte": 28
      }
    }
  }
}

POST /es_db/_search
{
  "query": {
    "range": {
      "age": {
        "gte": 25,
        "lte": 28
      }
    }
  },
  "from": 0,
  "size": 2,
  "_source": [
    "name",
    "age",
    "book"
  ],
  "sort": {
    "age": "desc"
  }
}
```

图灵课堂FOX

日期range

```
DELETE /product
POST /product/_bulk
{"index":{"_id":1}}
{"price":100,"date":"2021-01-01","productId":"XHDK-1293"}
{"index":{"_id":2}}
{"price":200,"date":"2022-01-01","productId":"KDKE-5421"}

GET /product/_mapping

GET /product/_search
{
  "query": {
    "range": {
      "date": {
        "gte": "now-2y"
      }
    }
  }
}
```

多id查询ids

ids 关键字 : 值为数组类型,用来根据一组id获取多个对应的文档

图灵课堂Fox

```
GET /es_db/_search
{
  "query": {
    "ids": {
      "values": [1,2]
    }
  }
}
```

模糊查询fuzzy

在实际的搜索中，我们有时候会打错字，从而导致搜索不到。在Elasticsearch中，我们可以使用fuzziness属性来进行模糊查询，从而达到搜索有错别字的情形。

fuzzy 查询会用到两个很重要的参数，fuzziness，prefix_length

- fuzziness：表示输入的关键字通过几次操作可以转变成为ES库里面的对应field的字段
-
  - 操作是指：新增一个字符，删除一个字符，修改一个字符，每次操作可以记做编辑距离为1，
  - 如中文集团到中威集团编辑距离就是1，只需要修改一个字符；
  - 该参数默认值为0，即不开启模糊查询。
  - 如果fuzziness值在这里设置成2，会把编辑距离为2的东东集团也查出来。
- prefix_length：表示限制输入关键字和ES对应查询field的内容开头的第n个字符必须完全匹配，不允许错别字匹配
-
  - 如这里等于1，则表示开头的字必须匹配，不匹配则不返回
  - 默认值也是0
  - 加大prefix_length的值可以提高效率和准确率。

```
GET /es_db/_search
{
  "query": {
    "fuzzy": {
      "address": {
        "value": "白运山",
        "fuzziness": 1
      }
    }
  }
}
```

注意: fuzzy 模糊查询 最大模糊错误 必须在0-2之间

- 搜索关键词长度为 2，不允许存在模糊
- 搜索关键词长度为3-5，允许1次模糊
- 搜索关键词长度大于5，允许最大2次模糊

高亮highlight

highlight 关键字: 可以让符合条件的文档中的关键词高亮。

highlight相关属性:

- pre_tags 前缀标签
- post_tags 后缀标签
- tags_schema 设置为styled可以使用内置高亮样式
- require_field_match 多字段高亮需要设置为false

图灵课堂Fox

示例数据

```
#指定ik分词器
PUT /products
{
  "settings" : {
      "index" : {
          "analysis.analyzer.default.type": "ik_max_word"
      }
  }
}

PUT /products/_doc/1
{
  "proId" : "2",
  "name" : "牛仔男外套",
  "desc" : "牛仔外套男装春季衣服男春装夹克修身休闲男生潮牌工装潮流头号青年春秋棒球服男 7705浅蓝常规 XL",
  "timestamp" : 1576313264451,
  "createTime" : "2019-12-13 12:56:56"
}

PUT /products/_doc/1
{
  "proId" : "6",
  "name" : "HLA海澜之家牛仔裤男",
  "desc" : "HLA海澜之家牛仔裤男2019时尚有型舒适HKNAD3E109A 牛仔蓝(A9)175/82A(32)",
  "timestamp" : 1576314265571,
```

```
  "createTime" : "2019-12-18 15:56:56"
}
```

测试

```
GET /products/_search
{
  "query": {
    "term": {
      "name": {
        "value": "牛仔"
      }
    }
  },
  "highlight": {
    "fields": {
      "*":{}
    }
  }
}
```

自定义高亮html标签

可以在highlight中使用pre_tags和post_tags

图灵课堂FOX

```
GET /products/_search
{
  "query": {
    "term": {
      "name": {
        "value": "牛仔"
      }
    }
  },
  "highlight": {
    "post_tags": ["</span>"],
    "pre_tags": ["<span style='color:red'>"],
    "fields": {
      "*":{}
    }
  }
}
```

多字段高亮

```
GET /products/_search
{
  "query": {
    "term": {
      "name": {
        "value": "牛仔"
      }
    }
  },
  "highlight": {
    "pre_tags": ["<font color='red'>"],
```

```
    "post_tags": ["<font/>"],
    "require_field_match": "false",
    "fields": {
      "name": {},
      "desc": {}
    }
  }
}
```

相关性和相关性算分

搜索是用户和搜索引擎的对话，用户关心的是搜索结果的相关性

- 是否可以找到所有相关的内容
- 有多少不相关的内容被返回了
- 文档的打分是否合理
- 结合业务需求，平衡结果排名

如何衡量相关性：

- Precision(查准率)—尽可能返回较少的无关文档
- Recall(查全率)–尽量返回较多的相关文档
- Ranking -是否能够按照相关度进行排序

相关性（Relevance）

搜索的相关性算分，描述了一个文档和查询语句匹配的程度。ES 会对每个匹配查询条件的结果进行算分 _score。

打分的本质是排序，需要把最符合用户需求的文档排在前面。ES 5之前，默认的相关性算分采用TF-IDF，现在采用BM 25。

图灵课堂FOX

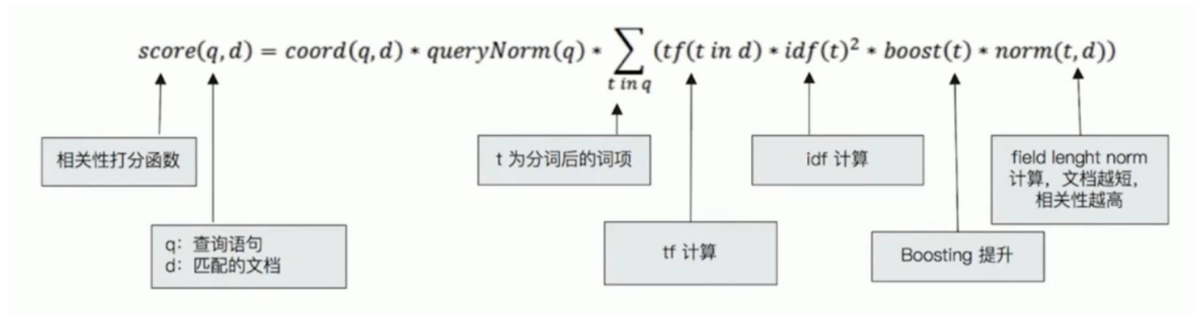
关键词	文档ID
JAVA	1,2,3
设计模式	1,2,3,4,5,6
多线程	2,3,7,9

什么是TF-IDF

TF-IDF（term frequency–inverse document frequency）是一种用于信息检索与数据挖掘的常用加权技术。

- TF-IDF被公认为是信息检索领域最重要的发明，除了在信息检索，在文献分类和其他相关领域有着非常广泛的应用。
- IDF的概念，最早是剑桥大学的“斯巴克.琼斯”提出
-
    - 1972年——“关键词特殊性的统计解释和它在文献检索中的应用”，但是没有从理论上解释IDF应该是用log(全部文档数/检索词出现过的文档总数)，而不是其他函数，也没有做进一步的研究
    - 1970，1980年代萨尔顿和罗宾逊，进行了进一步的证明和研究，并用香农信息论做了证明http://www.staff.city.ac.uk/~sb317/papers/foundations_bm25_review.pdf
- 现代搜索引擎，对TF-IDF进行了大量细微的优化

Lucene中的TF-IDF评分公式：



- **TF是词频(Term Frequency)**

检索词在文档中出现的频率越高，相关性也越高。

- **IDF是逆向文本频率(Inverse Document Frequency)**

每个检索词在索引中出现的频率，频率越高，相关性越低。

- **字段长度归一值（ field-length norm)**

字段的长度是多少？字段越短，字段的权重越高。检索词出现在一个内容短的 title 要比同样的词出现在一个内容长的 content 字段权重更大。

以上三个因素——词频（term frequency）、逆向文档频率（inverse document frequency）和字段长度归一值（field-length norm）——是在索引时计算并存储的，最后将它们结合在一起计算单个词在特定文档中的权重。

什么是BM25

- 从ES 5开始，默认算法改为BM 25
- 和经典的TF-IDF相比,当TF无限增加时，BM 25算分会趋于一个数值

- BM 25的公式

$$\mathrm{bm25}(d) = \sum_{t \in q, f_{t,d} > 0} \log\left(1 + \frac{N - df_t + 0.5}{df_t + 0.5}\right) \cdot \frac{f_{t,d}}{f_{t,d} + k \cdot \left(1 - b + b\frac{l(d)}{avgdl}\right)}$$

图灵课堂Fox

通过Explain API查看TF-IDF

示例：

```
PUT /test_score/_bulk
{"index":{"_id":1}}
{"content":"we use Elasticsearch to power the search"}
{"index":{"_id":2}}
{"content":"we like elasticsearch"}
{"index":{"_id":3}}
{"content":"Thre scoring of documents is caculated by the scoring formula"}
{"index":{"_id":4}}
{"content":"you know,for search"}

GET /test_score/_search
{
  "explain": true,
  "query": {
    "match": {
      "content": "elasticsearch"
    }
  }
}
```

Boosting Relevance

Boosting是控制相关度的一种手段。

参数boost的含义：

- 当boost > 1时，打分的相关度相对性提升
- 当0 < boost <1时，打分的权重相对性降低
- 当boost <0时，贡献负分

图灵课堂Fox

返回匹配positive查询的文档并降低匹配negative查询的文档相似度分。这样就可以在不排除某些文档的前提下对文档进行查询,搜索结果中存在只不过相似度分数相比正常匹配的要低;

```
GET /test_score/_search
{
  "query": {
    "boosting": {
      "positive": {
        "term": {
          "content": "elasticsearch"
        }
      },
      "negative": {
         "term": {
            "content": "like"
          }
      },
      "negative_boost": 0.2
    }
  }
}
```

应用场景：希望包含了某项内容的结果不是不出现，而是排序靠后。

布尔查询bool Query

一个bool查询,是一个或者多个查询子句的组合，总共包括4种子句，其中2种会影响算分，2种不影响算分。

- must: 相当于&& ，必须匹配，贡献算分
- should: 相当于|| ，选择性匹配，贡献算分
- must_not: 相当于! ，必须不能匹配，不贡献算分
- filter: 必须匹配，不贡献算法

在Elasticsearch中，有Query和 Filter两种不同的Context

- Query Context: 相关性算分
- Filter Context: 不需要算分(Yes or No) ,可以利用Cache，获得更好的性能

相关性并不只是全文本检索的专利，也适用于yes | no 的子句，匹配的子句越多，相关性评分

越高。如果多条查询子句被合并为一条复合查询语句，比如 bool查询，则每个查询子句计算得出的评分会被合并到总的相关性评分中。

bool查询语法

- 子查询可以任意顺序出现
- 可以嵌套多个查询
- 如果你的bool查询中，没有must条件,should中必须至少满足一条查询

```
GET /es_db/_search
{
  "query": {
    "bool": {
      "must": {
        "match": {
          "remark": "java developer"
        }
      },
      "filter": {
        "term": {
          "sex": "1"
        }
      },
      "must_not": {
        "range": {
          "age": {
            "gte": 30
          }
        }
      },
      "should": [
        {
          "term": {
            "address.keyword": {
              "value": "广州天河公园"
            }
          }
        },
        {
          "term": {
            "address.keyword": {
```

图灵课堂FOX

```
                "value": "广州白云山公园"
              }
            }
          }
        ],
        "minimum_should_match": 1
      }
    }
}
```

如何解决结构化查询“包含而不是相等”的问题

测试数据

```
POST /employee/_bulk
{"index":{"_id":1}}
{"name":"小明","interest":["跑步","篮球"]}
{"index":{"_id":2}}
{"name":"小红","interest":["跑步"]}
{"index":{"_id":3}}
{"name":"小丽","interest":["跳舞","唱歌","跑步"]}

POST /employee/_search
{
  "query": {
    "term": {
      "interest.keyword": {
        "value": "跑步"
      }
    }
  }
}
```

图灵课堂FOX

解决方案： 增加count字段，使用bool查询解决

- 从业务角度，按需改进Elasticsearch数据模型

```
POST /employee/_bulk
{"index":{"_id":1}}
{"name":"小明","interest":["跑步","篮球"],"interest_count":2}
{"index":{"_id":2}}
{"name":"小红","interest":["跑步"],"interest_count":1}
{"index":{"_id":3}}
{"name":"小丽","interest":["跳舞","唱歌","跑步"],"interest_count":3}
```

- 使用bool查询

```
# must 算分
POST /employee/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "term": {
            "interest.keyword": {
```

```
                "value": "跑步"
              }
            }
          },
          {
            "term": {
              "interest_count": {
                "value": 1
              }
            }
          }
        ]
      }
    }
}
# filter不算分
POST /employee/_search
{
  "query": {
    "bool": {
      "filter": [
        {
          "term": {
            "interest.keyword": {
              "value": "跑步"
            }
          }
        },
        {
          "term": {
            "interest_count": {
              "value": 1
            }
          }
        }
      ]
    }
  }
}
```

图灵课堂FOX

利用bool嵌套实现should not逻辑

```
GET /es_db/_search
{
  "query": {
    "bool": {
      "must": {
        "match": {
          "remark": "java developer"
        }
      },
      "should": [
        {
          "bool": {
            "must_not": [
```

```
              {
                "term": {
                  "sex": 1
                }
              }
            ]
          }
        }
      ],
      "minimum_should_match": 1
    }
  }
}
```

思考： 如何控制查询的精确度？

控制字段的Boosting

Boosting是控制相关度的一种手段。可以通过指定字段的boost值影响查询结果

参数boost的含义：

- 当boost > 1时，打分的相关度相对性提升
- 当0 < boost <1时，打分的权重相对性降低
- 当boost <0时，贡献负分

```
POST /blogs/_bulk
{"index":{"_id":1}}
{"title":"Apple iPad","content":"Apple iPad,Apple iPad"}
{"index":{"_id":2}}
{"title":"Apple iPad,Apple iPad","content":"Apple iPad"}

GET /blogs/_search
{
  "query": {
    "bool": {
      "should": [
        {
          "match": {
            "title": {
              "query": "apple,ipad",
              "boost": 1
            }
          }
        },
        {
          "match": {
            "content": {
              "query": "apple,ipad",
              "boost": 4
            }
          }
        }
      ]
    }
  }
}
```

图灵课堂Fox

案例：要求苹果公司的产品信息优先展示

```
POST /news/_bulk
{"index":{"_id":1}}
{"content":"Apple Mac"}
{"index":{"_id":2}}
{"content":"Apple iPad"}
{"index":{"_id":3}}
{"content":"Apple employee like Apple Pie and Apple Juice"}


GET /news/_search
{
  "query": {
    "bool": {
      "must": {
        "match": {
          "content": "apple"
        }
      }
    }
  }
}
```

利用must not 排除不是苹果公司产品的文档

```
GET /news/_search
{
  "query": {
    "bool": {
      "must": {
        "match": {
          "content": "apple"
        }
      },
      "must_not": {
        "match":{
          "content": "pie"
        }
      }
    }
  }
}
```

利用negative_boost降低相关性

- negative_boost 对 negative部分query生效
- 计算评分时,boosting部分评分不修改，negative部分query乘以negative_boost值
- negative_boost取值:0-1.0，举例:0.3

对某些返回结果不满意，但又不想排除掉（ must_not)，可以考虑boosting query的negative_boost。

```
GET /news/_search
{
```

图灵课堂Fox

```
  "query": {
    "boosting": {
      "positive": {
        "match": {
          "content": "apple"
        }
      },
      "negative": {
        "match": {
          "content": "pie"
        }
      },
      "negative_boost": 0.2
    }
  }
}
```

文档映射Mapping

Mapping类似数据库中的schema的定义，作用如下：

- 定义索引中的字段的名称
- 定义字段的数据类型，例如字符串，数字，布尔等
- 字段，倒排索引的相关配置(Analyzed or Not Analyzed,Analyzer)

ES中Mapping映射可以分为动态映射和静态映射。

动态映射：

在关系数据库中，需要事先创建数据库，然后在该数据库下创建数据表，并创建表字段、类型、长度、主键等，最后才能基于表插入数据。而Elasticsearch中不需要定义Mapping映射（即关系型数据库的表、字段等），在文档写入Elasticsearch时，会根据文档字段自动识别类型，这种机制称之为动态映射。

静态映射：

静态映射是在Elasticsearch中也可以事先定义好映射，包含文档的各字段类型、分词器等，这种方式称之为静态映射。

动态映射（Dynamic Mapping）的机制，使得我们无需手动定义Mappings，Elasticsearch会自动根据文档信息，推算出字段的类型。但是有时候会推算的不对，例如地理位置信息。当类型如果设置不对时，会导致一些功能无法正常运行，例如Range查询

Dynamic Mapping类型自动识别：

图灵课堂Fox

JSON类型	Elasticsearch 类型
字符串	• 匹配日期格式，设置成 Date • 配置数字设置为 float或者long，该选项默认关闭 • 设置为Text，并且增加 keyword 子字段
布尔值	boolean
浮点数	float
整数	long
对象	Object
数组	由第一个非空数值的类型所决定
空值	忽略

示例

```
#删除原索引
DELETE /user

#创建文档(ES根据数据类型, 会自动创建映射)
PUT /user/_doc/1
{
  "name":"fox",
  "age":32,
  "address":"长沙麓谷"
}

#获取文档映射
GET /user/_mapping
```

图灵课堂FOX

```
#动态映射

#删除原索引
DELETE /user

#创建文档(ES根据数据类型, 会自动创建映射)
PUT /user/_doc/1
{
  "name":"fox",
  "age":32,
  "address":"长沙麓谷"
}

#获取文档映射
GET /user/_mapping
```

```
1  {
2    "user" : {
3      "mappings" : {
4        "properties" : {
5          "address" : {
6            "type" : "text",
7            "fields" : {
8              "keyword" : {
9                "type" : "keyword",
10               "ignore_above" : 256
11             }
12           }
13         },
14         "age" : {
15           "type" : "long"
16         },
17         "name" : {
18           "type" : "text",
19           "fields" : {
20             "keyword" : {
21               "type" : "keyword",
22               "ignore_above" : 256
23             }
24           }
25         }
26       }
27     }
28   }
29 }
```

思考：能否后期更改Mapping的字段类型?

两种情况：

- 新增加字段
-
    - dynamic设为true时，一旦有新增字段的文档写入，Mapping 也同时被更新
    - dynamic设为false，Mapping 不会被更新，新增字段的数据无法被索引，但是信息会出现在_source中
    - dynamic设置成strict(严格控制策略)，文档写入失败，抛出异常

	true	false	strict
文档可索引	yes	yes	no
字段可索引	yes	no	no
Mapping被更新	yes	no	no

- 对已有字段，一旦已经有数据写入，就不再支持修改字段定义
-
    - Lucene 实现的倒排索引，一旦生成后，就不允许修改
    - 如果希望改变字段类型，必须 Reindex API，重建索引

原因：

- 如果修改了字段的数据类型，会导致已被索引的数据无法被搜索
- 但是如果是增加新的字段，就不会有这样的影响

图灵课堂FOX

测试

```
PUT /user
{
  "mappings": {
    "dynamic": "strict",
    "properties": {
      "name": {
        "type": "text"
      },
      "address": {
        "type": "object",
        "dynamic": "true"
      }
    }
  }
}
# 插入文档报错，原因为age为新增字段,会抛出异常
PUT /user/_doc/1
{
  "name":"fox",
  "age":32,
  "address":{
    "province":"湖南",
    "city":"长沙"
  }
}
```

dynamic设置成strict，新增age字段导致文档插入失败

修改dynamic后再次插入文档成功

```
#修改daynamic
PUT /user/_mapping
{
  "dynamic":true
}
```

对已存在的mapping映射进行修改

具体方法：

1）如果要推倒现有的映射，你得重新建立一个静态索引

2）然后把之前索引里的数据导入到新的索引里

3）删除原创建的索引

4）为新索引起个别名，为原索引名

图灵课堂Fox

```
POST _reindex
{
"source": {
"index": "user"
},
"dest": {
"index": "user2"
}
}

DELETE /user

PUT /user2/_alias/user

GET /user
```

注意：通过这几个步骤就实现了索引的平滑过渡,并且是零停机

常用Mapping参数配置

- index：控制当前字段是否被索引，默认为true。如果设置为false，该字段不可被搜索

```
DELETE /user
PUT /user
{
```

```
  "mappings" : {
      "properties" : {
        "address" : {
          "type" : "text",
          "index": false
        },
        "age" : {
          "type" : "long"
        },
        "name" : {
          "type" : "text"
        }
      }
    }
}

PUT /user/_doc/1
{
  "name":"fox",
  "address":"广州白云山公园",
  "age":30
}

GET /user

GET /user/_search
{
  "query": {
    "match": {
      "address": "广州"
    }
  }
}
```

```
PUT /user
{
  "mappings" : {
      "properties" : {
        "address" : {
          "type" : "text",
          "index": false
        },
        "age" : {
          "type" : "long"
        },
        "name" : {
          "type" : "text"
        }
      }
    }
}

GET /user/_search
{"query":{
  "match": {
    "address": "长沙"
  }
}}
```

设置为false,address不能被搜索

```
{
  "error" : {
    "root_cause" : [
      {
        "type" : "query_shard_exception",
        "reason" : "failed to create query: Cannot search on field [address] since it is not indexed.",
        "index_uuid" : "LkQZn60NTXmLhUtngGpFwA",
        "index" : "user"
      }
    ],
    "type" : "search_phase_execution_exception",
    "reason" : "all shards failed",
    "phase" : "query",
    "grouped" : true,
    "failed_shards" : [
      {
        "shard" : 0,
        "index" : "user",
        "node" : "6bUwgJ9USSubhnA9gs-_2A",
        "reason" : {
          "type" : "query_shard_exception",
          "reason" : "failed to create query: Cannot search on field [address] since it is not indexed.",
          "index_uuid" : "LkQZn60NTXmLhUtngGpFwA",
          "index" : "user",
          "caused_by" : {
            "type" : "illegal_argument_exception",
            "reason" : "Cannot search on field [address] since it is not indexed."
          }
        }
      }
    ]
  },
  "status" : 400
}
```

- 有四种不同基本的index_options配置，控制倒排索引记录的内容：
  - docs：记录doc id
  - freqs：记录doc id 和term frequencies（词频）
  - positions: 记录doc id / term frequencies / term position
  - offsets: doc id / term frequencies / term posistion / character offects

text类型默认记录postions，其他默认为 docs。记录内容越多，占用存储空间越大

图灵课堂Fox

```
DELETE /user
PUT /user
{
  "mappings" : {
      "properties" : {
        "address" : {
          "type" : "text",
          "index_options": "offsets"
        },
        "age" : {
          "type" : "long"
        },
        "name" : {
          "type" : "text"
        }
      }
    }
}
```

- null_value: 需要对Null值进行搜索，只有keyword类型支持设计Null_Value

```
DELETE /user
PUT /user
{
  "mappings" : {
      "properties" : {
        "address" : {
          "type" : "keyword",
          "null_value": "NULL"
        },
        "age" : {
          "type" : "long"
        },
        "name" : {
          "type" : "text"
        }
      }
    }
}

PUT /user/_doc/1
{
  "name":"fox",
  "address": null,
  "age":30

}

GET /user/_search
{
  "query": {
    "term": {
      "address": "NULL"
    }
  }
}
```

图灵课堂FOX

```
PUT /user
{
  "mappings" : {
      "properties" : {
        "address" : {
          "type" : "keyword",
          "null_value": "NULL"
        },
        "age" : {
          "type" : "long"
        },
        "name" : {
          "type" : "text"
        }
      }
    }
}

GET /user/_search
{
  "query": {
    "match": {
      "address": "NULL"
    }
  }
}
```

```
{
  "took" : 2,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 0.6931471,
    "hits" : [
      {
        "_index" : "user",
        "_type" : "_doc",
        "_id" : "1",
        "_score" : 0.6931471,
        "_source" : {
          "name" : "fox",
          "age" : 32,
          "address" : null
        }
      }
    ]
  }
```

- copy_to设置：将字段的数值拷贝到目标字段，满足一些特定的搜索需求。copy_to的目标字段不出现在_source中。

```
# 设置copy_to
PUT /address
{
  "mappings" : {
      "properties" : {
        "province" : {
          "type" : "keyword",
          "copy_to": "full_address"
        },
        "city" : {
          "type" : "text",
          "copy_to": "full_address"
        }
      }
    }
}

PUT /address/_doc/1
{
  "province": "湖南",
  "city": "长沙"
}
PUT /address/_doc/2
{
  "province": "湖南",
  "city": "常德"
}

GET /address/_search
{
  "query": {
    "match": {
      "full_address": {
```

图灵课堂FOX

```
        "query": "湖南常德",
        "operator": "and"
      }
    }
  }
}
```

Index Template

Index Templates可以帮助你设定Mappings和Settings，并按照一定的规则，自动匹配到新创建的索引之上

- 模版仅在一个索引被新创建时，才会产生作用。修改模版不会影响已创建的索引
- 你可以设定多个索引模版，这些设置会被"merge"在一起
- 你可以指定"order"的数值，控制"merging"的过程

```
PUT /_template/template_default
{
  "index_patterns": ["*"],
  "order": 0,
  "version": 1,
  "settings": {
    "number_of_shards": 1,
    "number_of_replicas": 1
  }
}


PUT /_template/template_test
{
  "index_patterns": ["test*"],
  "order": 1,
  "settings": {
    "number_of_shards": 1,
    "number_of_replicas": 1
  },
  "mappings": {
    "date_detection": false,  #关闭日期探测
    "numeric_detection": true
  }
}
```

图灵课堂FOX

Index Template的工作方式

当一个索引被新创建时：

- 应用Elasticsearch 默认的settings 和mappings
- 应用order数值低的Index Template 中的设定
- 应用order高的 Index Template 中的设定，之前的设定会被覆盖
- 应用创建索引时，用户所指定的Settings和 Mappings，并覆盖之前模版中的设定

```
#查看template信息
GET /_template/template_default
GET /_template/temp*

# 关闭日期探测，createDate会推断为text类型
PUT /testtemplate/_doc/1
```

```
{
  "orderNo": 1,
  "createDate": "2022/01/01"
}
GET /testtemplate/_mapping
GET /testtemplate/_settings

# 开启日期探测
PUT /testmy
{
  "mappings": {
    "date_detection": true
  }
}

PUT /testmy/_doc/1
{
  "orderNo": 1,
  "createDate": "2022/01/01"
}

GET /testmy/_mapping
```

Dynamic Template

图灵课堂Fox

根据Elasticsearch识别的数据类型，结合字段名称，来动态设定字段类型

- 所有的字符串类型都设定成Keyword，或者关闭keyword 字段
- is开头的字段都设置成 boolean
- long_开头的都设置成 long类型

```
PUT /my_test_index
{
  "mappings": {
    "dynamic_templates": [
      {
        "full_name":{
          "path_match": "name.*",
          "path_unmatch": "*.middle",
          "mapping":{
            "type": "text",
            "copy_to": "full_name"
          }

        }
      }

      ]
  }
}

PUT /my_test_index/_doc/1
{
  "name":{
    "first": "John",
```

```
    "middle": "Winston",
    "last": "Lennon"
  }
}


GET /my_test_index/_search
{
  "query": {
    "match": {
      "full_name": "John"
    }
  }
}
```

图灵课堂FOX