主讲老师：Fox

ES版本： v7.17.3

> 有道笔记地址： https://note.youdao.com/s/8BQL71Tg

1. 全文检索

1.1 什么是全文检索

全文检索是一种通过对文本内容进行全面索引和搜索的技术。它可以快速地在大量文本数据中查找包含特定关键词或短语的文档，并返回相关的搜索结果。全文检索广泛应用于各种信息管理系统和应用中，如搜索引擎、文档管理系统、电子邮件客户端、新闻聚合网站等。它可以帮助用户快速定位所需信息，提高检索效率和准确性。

> **查询**：有明确的搜索条件边界。比如，年龄 15~25 岁，颜色 = 红色，价格 < 3000，这里的 15、25、红色、3000 都是条件边界。即有明确的范围界定。
>
> **检索**：即全文检索，无搜索条件边界，召回结果取决于相关性，其相关性计算无明确边界性条件，如同义词、谐音、别名、错别字、混淆词、网络热梗等均可成为其相关性判断依据。

设想一个关于全文检索的场景，比如搜索Java设计模式：

id	标题	描述
1	Java中的23种设计模式	Java中23种设计模式，包括简单介绍,适用场景以及优缺点等
2	Java多线程设计模式	Java多线程与设计模式结合
3	设计模式之美	结合真实项目案例，从面向对象编程范式、设计原则、代码规范、重构技巧和设计模式5个方面详细介绍如何编写高质量代码。
4	JavaScript设计模式与开发实践	针对JavaScript语言特性全面介绍了更适合JavaScript程序员的了16个常用的设计模式
...	...	...
10亿	Java并发编程实战	深入浅出地介绍了Java线程和并发，是一本完美的Java并发参考手册
...	...	...

思考：用传统关系型数据库实现有什么问题？

如果是用MySQL存储文章，我们应该会使用这样的 SQL 去查询

```
select * from t_blog where content like "%Java设计模式%"
```

这种需要遍历所有的记录进行匹配，不但效率低，而且搜索结果不符合我们搜索时的期望。

1.2 全文检索的原理

在全文检索中，首先需要对文本数据进行处理，包括分词、去除停用词等。然后，对处理后的文本数据建立索引，索引会记录每个单词在文档中的位置信息以及其他相关的元数据，如词频、权重等。这个过程通常使用倒排索引（inverted index）来实现，倒排索引将单词映射到包含该单词的文档列表中，以便快速定位相关文档。

当用户发起搜索请求时，搜索引擎会根据用户提供的关键词或短语，在建立好的索引中查找匹配的文档。搜索引擎会根据索引中的信息计算文档的相关性，并按照相关性排序返回搜索结果。用户可以通过不同的搜索策略和过滤条件来精确控制搜索结果的质量和范围。

1.3 什么是倒排索引

正排索引（Forward Index）和倒排索引（Inverted Index）是全文检索中常用的两种索引结构，它们在索引和搜索的过程中扮演不同的角色。

正排索引（正向索引）

正排索引是将文档按顺序排列并进行编号的索引结构。每个文档都包含了完整的文本内容，以及其他相关的属性或元数据，如标题、作者、发布日期等。在正排索引中，可以根据文档编号或其他属性快速定位和访问文档的内容。正排索引适合用于需要对文档进行整体检索和展示的场景，但对于包含大量文本内容的数据集来说，正排索引的存储和查询效率可能会受到限制。

在MySQL 中通过 ID 查找就是一种正排索引的应用。

倒排索引（反向索引）

倒排索引是根据单词或短语建立的索引结构。它将每个单词映射到包含该单词的文档列表中。倒排索引的建立过程是先对文档进行分词处理，然后记录每个单词在哪些文档中出现，以及出现的位置信息。通过倒排索引，可以根据关键词或短语快速找到包含这些词语的文档，并确定它们的相关性。倒排索引适用于在大规模文本数据中进行关键词搜索和相关性排序的场景，它能够快速定位文档，提高搜索效率。

我们在创建文章的时候，建立一个关键词与文章的对应关系表，就可以称之为倒排索引。如下图所示：

关键词	文章ID	是否命中索引
Java	1,2	√
设计模式	1,2,3,4	√
多线程	2	
JavaScript	4	

2. ElasticSearch简介

2.1 ElasticSearch介绍

ElasticSearch（简称ES）是一个开源的分布式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，它专门设计用于处理大规模的文本数据和实现高性能的全文检索。

以下是一些 Elasticsearch 的特点和优势：

- 分布式架构：Elasticsearch 是一个分布式系统，可以轻松地水平扩展，处理大规模的数据集和高并发的查询请求。
- 全文检索功能：Elasticsearch 提供了强大的全文检索功能，包括分词、词项查询、模糊匹配、多字段搜索等，并支持丰富的查询语法和过滤器。
- 多语言支持：Elasticsearch 支持多种语言的分词器和语言处理器，可以很好地处理不同语言的文本数据。
- 高性能：Elasticsearch 使用倒排索引和缓存等技术，具有快速的搜索速度和高效的查询性能。

- 实时性：Elasticsearch 支持实时索引和搜索，可以几乎实时地将文档添加到索引中，并立即可见。
- 易用性：Elasticsearch 提供了简单易用的 RESTful API，方便进行索引管理、查询操作和数据分析。

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 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

2.2 ElasticSearch应用场景

只要用到搜索的场景，ES几乎都可以是最好的选择。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。

- 搜索引擎
- 站内搜索
- 日志管理与分析
- 大数据分析

2.3 技术选型

	Elasticsearch	Solr	MongoDB	MySQL
DB类型	搜索引擎	搜索引擎	文档数据库	关系型数据库
基于何种框架开发	Lucene	Lucene		
基于何种开发语言	Java	Java	C++	C、C++

数据结构	FST、Hash等			B+ Trees
数据格式	Json	Json/XML/CSV	Json	Row
分布式支持	原生支持	支持	原生支持	不支持
数据分区方案	分片	分片	分片	分库分表
业务系统类型	OLAP	OLAP	OLTP	OLTP
事务支持	不支持	不支持	多文档ACID事务	支持
数据量级	PB级	TB级~PB级	PB级	单库3000万
一致性策略	最终一致性	最终一致性	最终一致性即时一致性	即时一致性
擅长领域	海量数据全文检索大数据聚合分析	大数据全文检索	海量数据CRUD	强一致性ACID事务
劣势	不支持事务写入实时性低	海量数据的性能不如ES随着数据量的不断增大，稳定性低于ES	弱事务支持不支持join查询	大数据全文搜索性能低
查询性能	★★★★★	★★★★	★★★★★	★★★
写入性能	★★	★★	★★★★	★★★

3. ElasticSearch环境搭建

3.1 安装ElasticSearch

安装文档：https://www.elastic.co/guide/en/elasticsearch/reference/7.17/targz.html

温馨提示：初学者建议直接安装windows版本的elasticSearch，安装包解压后就可以直接运行

windows安装ElasticSearch

1）下载ElasticSearch

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

选择版本：7.17.3

```
# windows
https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-7.17.3-windows-x86_64.zip
```

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	7.x 以后特有，自带的 java 环境，8.x版本自带 jdk 17
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

2）配置JDK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
- 运行Elasticsearch，需安装并配置JDK。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的JDK环境变量生效的优先级配置顺序 ES_JAVA_HOME>JAVA_HOME>ES_HOME
  - ES_JAVA_HOME：这个环境变量用于指定Elasticsearch使用的Java运行时环境的路径。在启动Elasticsearch时，它会检查ES_JAVA_HOME环境变量并使用其中的Java路径。
  - ES_HOME：这个环境变量指定Elasticsearch的安装路径。它用于定位Elasticsearch的配置文件、插件和其他相关资源。设置ES_HOME环境变量可以让您在命令行中更方便地访问Elasticsearch的目录结构和文件。

可以参考es的环境文件elasticsearch-env.bat

windows下，设置ES_JAVA_HOME和ES_HOME的环境变量

3）启动ElasticSearch服务

进入bin目录，直接运行elasticsearch.bat

测试，浏览器中访问：http://localhost:9200/

centos7安装ElasticSearch

centos7中ES环境搭建视频：https://pan.baidu.com/s/1PsTNbpDy--M-pvFWb3aehQ?pwd=nwxl

1）下载ElasticSearch

```
# centos7
wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-7.17.3-linux-x86_64.tar.gz
```

2）配置JDK环境

```
# linux  进入用户主目录，比如/home/es目录下，设置用户级别的环境变量
vim .bash_profile
#设置ES_JAVA_HOME和ES_HOME的路径
export ES_JAVA_HOME=/home/es/elasticsearch-7.17.3/jdk/
export ES_HOME=/home/es/elasticsearch-7.17.3
#执行以下命令使配置生效
source .bash_profile
```

3) 配置ElasticSearch

修改elasticsearch.yml配置

```
vim elasticsearch.yml

#开启远程访问
network.host: 0.0.0.0
#单节点模式   初学者建议设置为此模式
discovery.type: single-node
```

ElasticSearch配置参数

参考： https://www.elastic.co/guide/en/elasticsearch/reference/7.17/important-settings.html

- 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。不要在不同的环境中重用相同的集群名称，否则，节点可能会加入错误的集群。

- 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

节点对外提供服务的地址以及集群内通信的ip地址，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。

- http.port

配置当前ES节点对外提供服务的http端口，默认 9200

- transport.port：

节点通信端口号，默认 9300

- 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。

4) 配置JVM参数

修改config/jvm.options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

5）启动ElasticSearch服务

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用户启动
bin/elasticsearch

# -d 后台启动
bin/elasticsearch -d
```

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
```

6）启动ES服务常见错误解决方案

如果ES服务启动异常，会有提示：

[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"]

#或者指定配置单节点（集群单节点）
discovery.type: single-node
```

3.2 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。

参考文档：https://www.elastic.co/guide/en/kibana/7.17/get-started.html

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

```
#windows
https://artifacts.elastic.co/downloads/kibana/kibana-7.17.3-windows-x86_64.zip
#linux
wget https://artifacts.elastic.co/downloads/kibana/kibana-7.17.3-linux-x86_64.tar.gz
```

2）修改Kibana.yml

```
vim config/kibana.yml

server.port: 5601   #指定Kibana服务器监听的端口号
server.host: "localhost"  #指定Kibana服务器绑定的主机地址
elasticsearch.hosts: ["http://localhost:9200"]  #指定Kibana连接到的Elasticsearch实例的访问地址
i18n.locale: "zh-CN"   #将 Kibana 的界面语言设置为简体中文
```

3）运行Kibana

windows

直接执行kibana.bat

Linux

注意：kibana也需要非root用户启动

```
bin/kibana
#后台启动
nohup  bin/kibana &

#查询kibana进程
netstat -tunlp | grep 5601
```

4）访问Kibana: http://localhost:5601/app/dev_tools#/console

cat API

cat API 是 Elasticsearch 提供的一个用于查看和显示集群信息的 RESTful API。它可以用于获取关于索引、节点、分片、健康状态等各种集群相关的信息。

```
/_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
```

3.3 Elasticsearch安装分词插件

Elasticsearch提供插件机制对系统进行扩展，以安装分词插件为例：

在线安装analysis-icu分词插件

```
#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

```
# _analyzer API可以用来查看指定分词器的分词结果
POST _analyze
```

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

离线安装ik中文分词插件

本地下载elasticsearch-analysis-ik-7.17.3.zip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。

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

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"
}

```

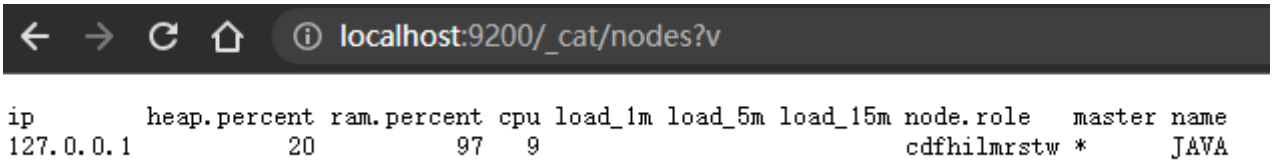
4. ElasticSearch快速开始

4.1 ElasticSearch核心概念

节点：Node

一个节点就是一个Elasticsearch的实例，可以理解为一个 ES 的进程。

注意：一个节点 ≠ 一台服务器



角色：Roles

ES的角色分类：

- 主节点（active master）：一般指活跃的主节点，一个集群中只能有一个，主要作用是对集群的管理。
- 候选节点（master-eligible）：当主节点发生故障时，参与选举，也就是主节点的替代节点。
- 数据节点（data node）：数据节点保存包含已编入索引的文档的分片。数据节点处理数据相关操作，如CRUD、搜索和聚合。这些操作是 I/O 密集型、内存密集型和 CPU 密集型的。监控这些资源并在它们过载时添加更多数据节点非常重要。
- 预处理节点（ingest node）：预处理节点有点类似于logstash的消息管道，所以也叫ingest pipeline，常用于一些数据写入之前的预处理操作。

注意：如果 node.roles 为缺省配置，那么当前节点具备所有角色。

索引：Index

索引在 ES 中所表述的含义和 MySQL 中的索引完全不同，在 MySQL 中索引指的是加速数据查询的一种特殊的数据结构，如 normal index。

而在 ES 中，索引表述的含义等价于 MySQL 中的表（仅针对 ES 7.x 以后版本），注意这里只是类比去理解，索引并不等于表。

索引的组成部分：

- alias：索引别名
- settings：索引设置，常见设置如分片和副本的数量等。
- mapping：映射，定义了索引中包含哪些字段，以及字段的类型、长度、分词器等。

在 ES 中，索引在不同的特定条件下可以表示三种不同的意思：

- 表示源文件数据：当做数据的载体，即类比为数据表，通常称作 index 。例如：通常说 集群中有 product 索引，即表述当前 ES 的服务中存储了 product 这样一张“表”。
- 表示索引文件：以加速查询检索为目的而设计和创建的数据文件，通常承载于某些特定的数据结构，如哈希、

FST 等。例如：通常所说的 正排索引 和 倒排索引（也叫正向索引和反向索引）。就是当前这个表述，索引文件和源数据是完全独立的，索引文件存在的目的仅仅是为了加快数据的检索，不会对源数据造成任何影响，

- 表示创建数据的动作：通常说创建或添加一条数据，在 ES 的表述为索引一条数据或索引一条文档，或者 index 一个 doc 进去。此时索引一条文档的含义为向索引中添加数据。

类型：Type（ES 7.x 之后版本已删除此概念）

在较早的ES版本中，索引可以包含多个类型，每个类型代表了不同的文档结构。然而，从ES 7.x版本开始，类型已经被弃用，一个索引只能包含一个文档类型。

ES 7.x

- 不推荐在请求中指定类型。例如，索引文档不再需要文档type。新的索引 API 适用PUT {index}/_doc/{id}于显式 ID 和POST {index}/_doc 自动生成的 ID。请注意，在 7.0 中，_doc是路径的永久部分，表示端点名称而不是文档类型。
- 索引创建、索引模板和映射 API 中的include_type_name参数将默认为false. 完全设置参数将导致弃用警告。
- _default_映射类型被删除 。

ES 8.x

- 不再支持在请求中指定类型。
- 该include_type_name参数被删除。

文档：Document

文档是ES中的最小数据单元。它是一个具有结构化JSON格式的记录。文档可以被索引并进行搜索、更新和删除操作。

文档元数据，所有字段均以下划线开头，为系统字段，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。

4.2 ElasticSearch索引操作

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

创建索引

格式: PUT /索引名称

索引命名规范：

- 以小写英文字母命名索引
- 不要使用驼峰命名法则
- 如过出现多个单词的索引名称，以全小写 + 下划线分隔的方式：如test_index。

ES 索引创建成功之后，以下属性将不可修改

- 索引名称
- 主分片数量
- 字段类型

```
#创建索引
PUT /es_db
```

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
```

删除索引

格式: DELETE /索引名称

```
DELETE /es_db
```

设置 Settings

创建索引的时候指定 settings

```
PUT <index_name>
{
  "settings": {}
}
```

创建索引时可以设置分片数和副本数

```
#创建索引es_db，指定其主分片数量为 3，每个主分片的副本数量为 2
PUT /es_db
{
    "settings" : {
        "number_of_shards" : 3,
        "number_of_replicas" : 2
    }
}
```

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"
        }
    }
}
```

静态索引设置

只能在创建索引时或在关闭状态的索引上设置。

- index.number_of_shards：索引的主分片的个数，默认为 1，此设置只能在创建索引时设置

动态索引设置

即可以使用 _setting API 在实时修改的配置项。

- index.number_of_replicas：每个主分片的副本数。默认为 1，允许配置为 0。
- index.refresh_interval：执行刷新操作的频率，默认为1s. 可以设置 -1 为禁用刷新。
- index.max_result_window：from + size 搜索此索引 的最大值，默认为 10000。

使用 _setting 只能修改允许动态修改的配置项

```

#修改索引配置，把每个主分片的副本数量修改为 1
PUT /es_db/_settings
{
    "index" : {
        "number_of_replicas" : 1
    }
}
```

设置文档映射Mapping

ES 中的 mapping 有点类似与关系数据库中表结构的概念，在 MySQL 中，表结构里包含了字段名称，字段的类型还有索引信息等。在 Mapping 里也包含了一些属性，比如字段名称、类型、字段使用的分词器、是否评分、是否创建索引等属性，并且在 ES 中一个字段可以有多个类型。ES中Mapping可以分为动态映射和静态映射。

```

查看完整的索引 mapping
GET /<index_name>/_mappings

查看索引中指定字段的 mapping
```

```
GET /<index_name>/_mappings/field/<field_name>
```

mapping 的使用禁忌

- ES 没有隐式类型转换
- ES 不支持类型修改
- 生产环境尽可能的避免使用 **动态映射**（dynamic mapping）

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，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。

自动类型推断规则

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
```

静态映射

静态映射也叫做显式映射，即：在索引文档写入之前，人为创建索引并且指定索引中每个字段类型、分词器等参数。

```
PUT /user
{
  "settings": {
    "number_of_shards": "1",
    "number_of_replicas": "1"
  },
  "mappings": {
    "properties": {
      "name": {
        "type": "keyword"
      },
      "age" : {
          "type" : "long"
      },
      "address" : {
          "type" : "text"
      }
    }
  }
}
```

常用Mapping参数配置

参数名称	释义
analyzer	**指定分析器，只有 text 类型字段支持。**
copy_to	该参数允许将多个字段的值复制到组字段中，然后可以将其作为单个字段进行查询
dynamic	控制是否可以动态添加新字段，支持以下四个选项： **true**：（默认）允许动态映射 **false**：忽略新字段。这些字段不会被索引或搜索，但仍会出现在_source返回的命中字段中。这些字段不会添加到映射中，必须显式添加新字段。 **runtime**：新字段作为运行时字段添加到索引中，这些字段没有索引，是_source在查询时加载的。

	strict：如果检测到新字段，则会抛出异常并拒绝文档。必须将新字段显式添加到映射中。
doc_values	为了提升排序和聚合效率，默认true，如果确定不需要对字段进行排序或聚合，也不需要通过脚本访问字段值，则可以禁用doc值以节省磁盘空间（不支持 text 和 annotated_text）
eager_global_ordinals	用于聚合的字段上，优化聚合性能。
enabled	是否创建倒排索引，可以对字段操作，也可以对索引操作，如果不创建索引，任然可以检索并在_source元数据中展示，谨慎使用，该状态无法修改。
fielddata	查询时内存数据结构，在首次用当前字段聚合、排序或者在脚本中使用时，需要字段为fielddata数据结构，并且创建倒排索引保存到堆中
fields	给 field 创建多字段，用于不同目的（全文检索或者聚合分析排序）

format	用于格式化代码，如
	`"data":{`
	`"type": "data",`
	`"format": "yyyy-MM-dd HH:mm:ss||yyyy-MM-dd||epoch_millis"`
	`}`

index	是否对创建对当前字段创建倒排索引，默认true，如果不创建索引，该字段不会通过索引被搜索到,但是仍然会在 source 元数据中展示。
norms	是否禁用评分（在filter和聚合字段上应该禁用）
null_value	为 null 值设置默认值
search_analyzer	设置单独的查询时分析器

示例：

- 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"
    }
  }
}
```

- 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设置成strict(严格控制策略)，文档写入失败，抛出异常

```
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
}
```

注意：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，可以利用 reindex API，重建索引

使用ReIndex重建索引

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

通过这几个步骤可以实现了索引的平滑过渡,并且是零停机

```
# 1. 重新建立一个静态索引
PUT /user2
{
  "mappings": {
    "properties": {
      "name": {
        "type": "text"
      },
      "address": {
        "type": "text",
        "analyzer": "ik_max_word"
      }
    }
  }
}
# 2. 把之前索引里的数据导入到新的索引里
POST _reindex
{
"source": {
"index": "user"
},
"dest": {
"index": "user2"
}
}
# 3. 删除原创建的索引
DELETE /user
# 4. 为新索引起个别名，为原索引名
PUT /user2/_alias/user

```

```
GET /user
```

4.3 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"
}
```

索引文档

- 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"
}
```

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

create: 如果ID已经存在，会失败

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

URI Query

```
#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
```

DSL Query

DSL（Domain Specific Language领域专用语言）查询是使用Elasticsearch的查询语言来构建查询的方式。

DSL Query会在后续课程中详细讲解

```

```

```
# match 匹配查询，会对查询文本分词后匹配
GET /es_db/_search
{
  "query": {
    "match": {
      "address": "广州白云"
    }
  }
}

# term 词项查询，属于精确查询，不会对查询文本分词
# 思考：能否查到文档？
GET /es_db/_search
{
  "query": {
    "term": {
      "address": "广州白云"
    }
  }
}
```

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
```

- 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"
  }
}
```

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：

删除文档

格式: DELETE /索引名称/_doc/id

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

4.4 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：

```
{"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

```
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}
```

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}}
```

组合应用

```
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}
{"delete":{"_index":"article", "_type":"_doc", "_id":3}}
{"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"]
 }
```

_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" : {}}}
```

4.5 Spring Boot整合ElasticSearch实战

官方文档：https://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html/#new-features.4-4-0

版本选择

Elasticsearch 7.17.3 对应依赖 Spring Data Elasticsearch 4.4.x，对应springboot版本2.7.x

Spring Data Release Train	Spring Data Elasticsearch	Elasticsearch	Spring Framework	Spring Boot
2022.0 (Raj)	4.4.x	7.17.3	5.3.x	2.7.x
2021.1 (Q)	4.3.x	7.15.2	5.3.x	2.6.x
2021.0 (Pascal)	4.2.x	7.12.0	5.3.x	2.5.x
2020.0 (Ockham)[1]	4.1.x[1]	7.9.3	5.3.2	2.4.x
Neumann[1]	4.0.x[1]	7.6.2	5.2.12	2.3.x
Moore[1]	3.2.x[1]	6.8.12	5.2.12	2.2.x
Lovelace[1]	3.1.x[1]	6.2.2	5.1.19	2.1.x
Kay[1]	3.0.x[1]	5.5.0	5.0.13	2.0.x

引入依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-elasticsearch</artifactId>
</dependency>
```

yml配置

```yaml
spring:
  elasticsearch:
    uris: http://localhost:9200
    connection-timeout: 3s
```

创建实体

```java
@Data
@AllArgsConstructor
@Document(indexName = "employees")
public class Employee {
    @Id
    private Long id;
    @Field(type= FieldType.Keyword)
    private String name;
    private int sex;
    private int age;
    @Field(type= FieldType.Text,analyzer="ik_max_word")
    private String address;
    private String remark;
}
```

实现ElasticsearchRepository

该接口是框架封装的用于操作Elastsearch的高级接口

```java
@Repository
public interface EmployeeRepository extends ElasticsearchRepository<Employee, Long> {
```

```
    List<Employee> findByName(String name);
}
```

测试

```
@Autowired
EmployeeRepository employeeRepository;

@Test
public void testDocument(){

    Employee employee = new Employee(1L,"fox666",1,32,"长沙麓谷","java architect");
     //插入文档
    employeeRepository.save(employee);

    //根据id查询
     Optional<Employee> result = employeeRepository.findById(1L);
     log.info(String.valueOf(result.get()));

     //根据name查询
     List<Employee> list = employeeRepository.findByName("fox666");
     log.info(String.valueOf(list.get(0)));

}
```

使用ElasticsearchRestTemplate

ElasticsearchRestTemplate模板类，封装了便捷操作Elasticsearch的模板方法，包括 索引 / 映射 / CRUD 等底层操作和高级操作。

```
@Autowired
ElasticsearchRestTemplate elasticsearchRestTemplate;
```

索引操作

```java
@Test
public void testCreateIndex(){
    //创建索引
    IndexOperations indexOperations = elasticsearchRestTemplate.indexOps(IndexCoordinates.of("employee_index"));
    if (indexOperations.exists()) {
        log.info("索引已经存在");
    }else {
        //创建索引
        indexOperations.create();
    }
}
@Test
public void testDeleteIndex(){
    //删除索引
    IndexOperations indexOperations = elasticsearchRestTemplate.indexOps(IndexCoordinates.of("employee_index"));
    indexOperations.delete();
}
```

文档操作

```java
@Test
public void testQueryDocument(){
    NativeSearchQueryBuilder builder = new NativeSearchQueryBuilder();
    //查询
    builder.withQuery(QueryBuilders.matchQuery("address","公园"));
    // 设置分页信息
    builder.withPageable(PageRequest.of(0, 5));
    // 设置排序
    builder.withSort(SortBuilders.fieldSort("age").order(SortOrder.DESC));

    SearchHits<Employee> search = elasticsearchRestTemplate.search(builder.build(), Employee.class);
    List<SearchHit<Employee>> searchHits = search.getSearchHits();
    for (SearchHit hit: searchHits){
        log.info("返回结果: "+hit.toString());
```

```
    }

}


@Test
public void testInsertBatch(){
    List<Employee> employees = new ArrayList<>();
    employees.add(new Employee("2","张三",1,25,"广州天河公园","java developer"));
    employees.add(new Employee("3","李四",1,28,"广州荔湾大厦","java assistant"));
    employees.add(new Employee("4","小红",0,26,"广州白云山公园","php developer"));

    List<IndexQuery> queries = new ArrayList<>();
    for (Employee employee : employees) {
        IndexQuery indexQuery = new IndexQuery();
        indexQuery.setId(employee.getId());
        String json = JSONObject.toJSONString(employee);
        indexQuery.setSource(json);
        queries.add(indexQuery);
    }
    //bulk批量插入
    elasticsearchRestTemplate.bulkIndex(queries,Employee.class);
}
```