



中国矿业大学
CHINA UNIVERSITY OF MINING AND TECHNOLOGY

本科生课程报告（论文）

ElasticSearch 体系架构与相关原理 ElasticSearch Architecture and Associated Principles

作 者：郭子杨
老 师：闫秋艳副教授

中国矿业大学
2020 年 11 月

中国矿业大学

本科生课程报告（论文）

ElasticSearch 体系架构与相关原理
ElasticSearch Architecture and Associated
Principles

作 者	郭子杨	学 号	08173042
导 师	闫秋艳	职 称	副教授
学 院	计算机科学与技术学院	专 业	计科 17-06

二〇二〇年十一月

ElasticSearch 体系架构与相关原理

ElasticSearch Architecture and Associated Principles

课程名称 《分布式数据库》

报告时间 2020 年 11 月 2 日

姓 名 郭子杨

学 号 08173042

专 业 计算机科学与技术

任课教师 闫秋艳

摘 要

随着互联网的不断拓展,海量的数据访问和处理造成传统的集中式数据库开始表现出性能瓶颈,分布式数据库的研究和场景使用应运而生,其中 ElasticSearch 的产生为解决复杂条件下的数据查询提供了新的解决方案。本文从 ElasticSearch 的产生背景说起,介绍了 ElasticSearch 的定义与应用场景,分析了其系统架构,深入研究了写入与查询的原理,在和 HBase 这类分布式数据库的对比中阐明了 ElasticSearch 的优缺点,最后以一个实例做结尾,体现了具体的实践步骤。

关键词: 分布式数据库, 关系数据库, ElasticSearch 系统架构, ElasticSearch 查询原理, 倒排索引

Abstract

With the continuous expansion of the Internet, traditional centralized databases start to show performance bottlenecks due to massive data access and processing. As a result, research and scenario use of distributed databases emerge as The Times require. ElasticSearch provides a new solution to solve data query under complex conditions. Starting from the background of ElasticSearch generation, this paper introduces the definition and application scenario of ElasticSearch, analyzes its system architecture, makes an in-depth study of the principle of writing and query, illustrates the advantages and disadvantages of ElasticSearch in comparison with distributed databases such as HBase, and finally ends with an example to show specific practical steps.

Keyword: Distributed database, relational Database, ElasticSearch Architecture, ElasticSearch Search Principle, Inversion Index

目 录

摘 要	I
目 录	II
1 Elasticsearch 简介	1
1.1 产生背景 (Background)	1
1.2 定义 (Definition)	1
1.3 场景选型对比 (Definition)	1
2 Elasticsearch 体系架构	3
2.1 基本概念 (Basic Concept)	3
2.2 工作原理 (Operating Principle)	4
3 Elasticsearch 写入、查询的原理	6
3.1 ES 写入数据原理 (ElasticSearch Writes Data Procedure)	6
3.2 ES 查询数据原理 (ElasticSearch Searches Data Procedure)	7
4 Elasticsearch 与 HBase 的比较	11
4.1 数据存储方式 (ElasticSearch Writes Data Procedure)	11
4.2 容灾方式 (Disaster Recovery Mode)	11
4.3 查询方式 (Search Mode)	12
4.4 优缺点总结 (Summary of Advantages and Disadvantages)	12
5 一个简单的 Elasticsearch 实例	12
参考文献	15
附录	16

1 Elasticsearch 简介

1 Brief Introduction to Elasticsearch

1.1 产生背景 (Background)

对于门户网站，往往会为用户提供关键字检索功能，比如百度的搜索功能，检索网页时，百度后台就会按照关键字进行查找，然后按照权重来进行从上向下的排序，将其高亮展示。

再比如京东或者淘宝，搜索商品时，如果需要高精度展示搜索的商品，就要求根据关键词进行海量数据的快速的检索，这通常需要先对关键字进行分词，再根据这些词去海量的数据中检索，然后根据权重把检索出来的信息进行排序展示。

当数据不断膨胀时，传统数据库单表查询能力即便经过了优化，也会经常出现查询超时的现象，通过对数据库的横向与纵向扩容，对表的横向纵向拆分，可以在一定程度上提高查询速度，但同时也带来单点故障频繁、维护难度大等问题。

于是产生了专精于搜索引擎的 Elasticsearch。

与此同时，正如《Elasticsearch 权威指南》的作者 Shay Banon 在 2017 年的阿里巴巴云栖大会上所说的那样“原始数据如果只是躺在磁盘里面根本就毫无用处。Elasticsearch 真正强大之处在于可以从无规律的数据中找出有意义的信息——从‘大数据’到‘大信息’”^[1]。只要数据存在于 Elasticsearch，这些数据的被探索力就比放在数据库里的强，你随时可以在里面挖掘出商机。而这一切的分析数据的能力，都是建立在快速的查询上的，如果没有快速的查询，分析能力无从谈起。

1.2 定义 (Definition)

ElasticSearch 既是搜索引擎，又是数据库^[2]。

Elasticsearch 在搜索引擎数据库领域排名绝对第一，内核基于 Lucene 构建，支持全文搜索是职责所在，提供了丰富友好的 API。使用 Json 格式来承载数据模型，已经成为事实上的文档型数据库，这一方面与 MongoDB 类似。

Elasticsearch 通常简称为 ES，与名为 Logstash 的数据收集和日志解析引擎以及名为 Kibana 的分析和可视化平台一起开发的。这三个产品被设计成一个集成解决方案，称为 ELK。

1.3 场景选型对比 (Scenario Selection Comparison)

一般来说，分布式数据可分为大致三类：

- (1) 支持持久化存储的分布式存储系统，如 MySQL，OceanBase。
- (2) 偏向于计算的分布式计算框架，如 Hadoop HDFS，ES。
- (3) 分布式消息队列，如 Redis，Kafka。

表 1-1 以 Elasticsearch、Redis、MySQL 分布式集群、MongoDB 四个分布式

数据库为例，进行了应用场景对比^[3]。

分布式数据库	应用场景
ES	(1) 分布式的搜索引擎和数据分析引擎,全文检索 (2) 对海量数据进行近实时的处理，站内搜索，系统搜索，数据分析
Redis	(1) 常规计数 (2) 用户信息变更 (3) 缓存处理,作为主数据库的缓存 (4) 队列系统
MySQL 分布式集群	(1) 多库冗余 (2) 水平切分表 (3) 自定义分库分表方案 (4) 分布式管理器
MongoDB	(1) 高度伸缩性数据 (2) 用作缓存 (3) 大尺寸、低价值的数据 (4) 高伸缩性的场景 (5) 用于对象及 JSON 数据的存储

表 1-1 分布式数据库场景选型对比

Table 1-1 Selection and Comparison of Distributed Database Scenarios

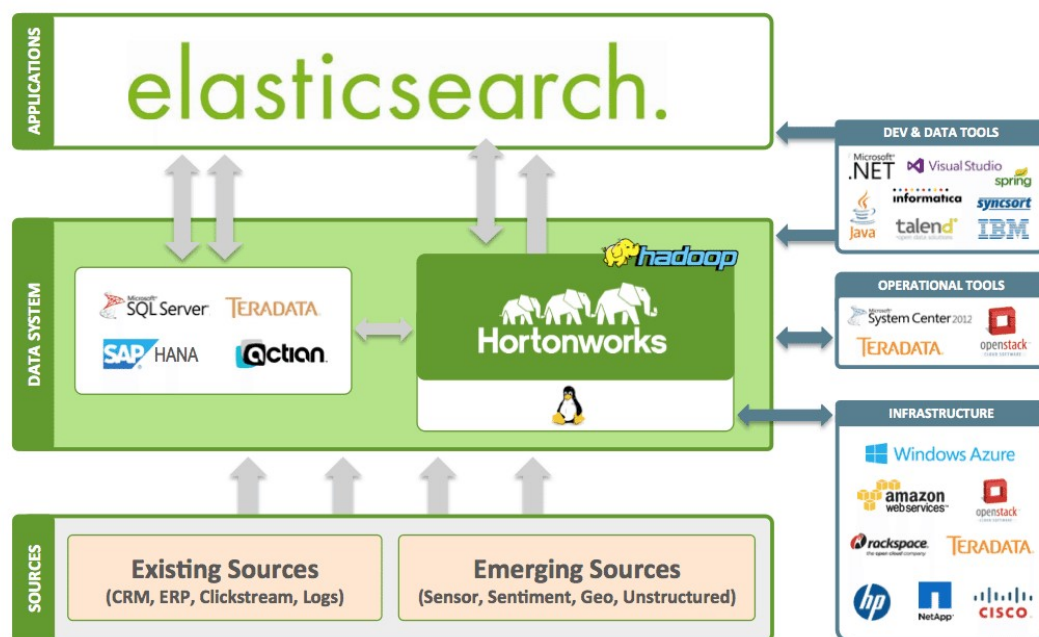


图 1-1 ElasticSearch 的典型场景

Figure 1-1 A Typical Scenario of ElasticSearch

2 Elasticsearch 体系架构

2 Elasticsearch Architecture

2.1 基本概念（Basic Concept）

首先说明 Elasticsearch 中几个关键概念：

节点（Node）：物理概念，一般是一台机器上的一个进程。

索引（Index），逻辑概念，其中包括配置信息 mapping 和倒排正排数据文件，一个索引的数据文件可能会分布于一台机器，也有可能分布于多台机器。

分片（Shard）：为了支持更大量的数据，索引一般会按某个维度分成多个部分，每个部分就是一个分片，分片被节点(Node)管理。一个节点(Node)一般会管理多个分片，这些分片可能是属于同一份索引，也有可能属于不同索引，但是为了可靠性和可用性，同一个索引的分片尽量会分布在不同节点(Node)上。分片有两种，主分片和副本分片。

副本（Replica）：同一个分片(Shard)的备份数据，一个分片可能会有 0 个或多个副本，这些副本中的数据保证强一致或最终一致。

图 1-2 中介绍了一个 Index 的示例。

Index 1：蓝色部分，有 3 个 shard，分别是 P1，P2，P3，位于 3 个不同的 Node 中，这里没有 Replica。

Index 2：绿色部分，有 2 个 shard，分别是 P1，P2，位于 2 个不同的 Node 中。并且每个 shard 有一个 replica，分别是 R1 和 R2。基于系统可用性的考虑，同一个 shard 的 primary 和 replica 不能位于同一个 Node 中。这里 Shard1 的 P1 和 R1 分别位于 Node3 和 Node2 中，如果某一刻 Node2 发生宕机，服务基本不会受影响，因为还有一个 P1 和 R2 都还是可用的。因为是主备架构，当主分片发生故障时，需要切换，这时候需要选举一个副本作为新主，这里除了会耗费一点点时间外，也会有丢失数据的风险。

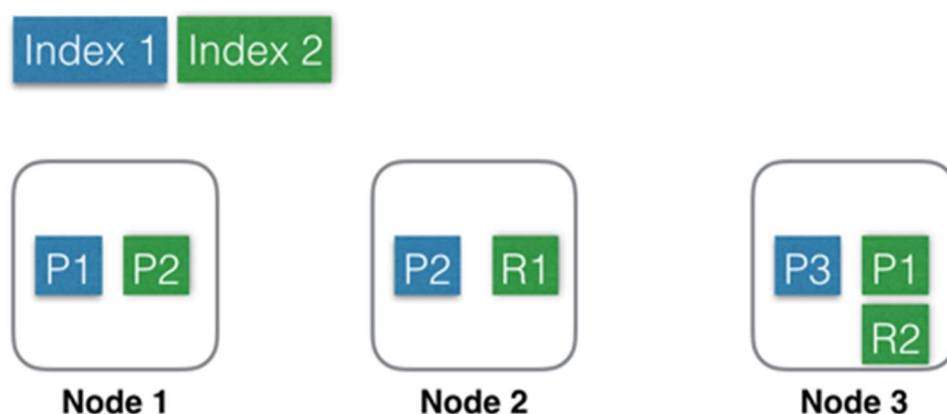


图 2-1 一个 Index 的示例

Figure 2-1 A Sample of Index

2.2 工作原理（Operating Principle）

ES 中存储数据的基本单位是索引，如果要在 ES 中存储一些订单数据，只需在 ES 中创建一个索引 `order_idx`，所有的订单数据就都写到这个索引里面去，一个索引相当于 MySQL 里的一张表。一个 `index` 里可以有多个 `type`，每个 `type` 的字段都是差不多的，但是有一些细微的差别。很多情况下，一个 `index` 里可能就是一个 `type`，如果一个 `index` 里有多个 `type` 的话可以认为 `index` 是一个类别的表，具体的每个 `type` 代表了 MySQL 中的一张表。每个 `type` 有一个 `mapping`，如果认为一个 `type` 是具体的一张表，`index` 就代表多个 `type` 同属于的一个类型，而 `mapping` 就是这个 `type` 的表结构定义。向 `index` 里的一个 `type` 里写的一条数据，叫做 `document`，一条 `document` 就代表了 MySQL 中某个表里的一行，每个 `document` 有多个 `field`，`field` 就是 `document` 中的一个字段的值。

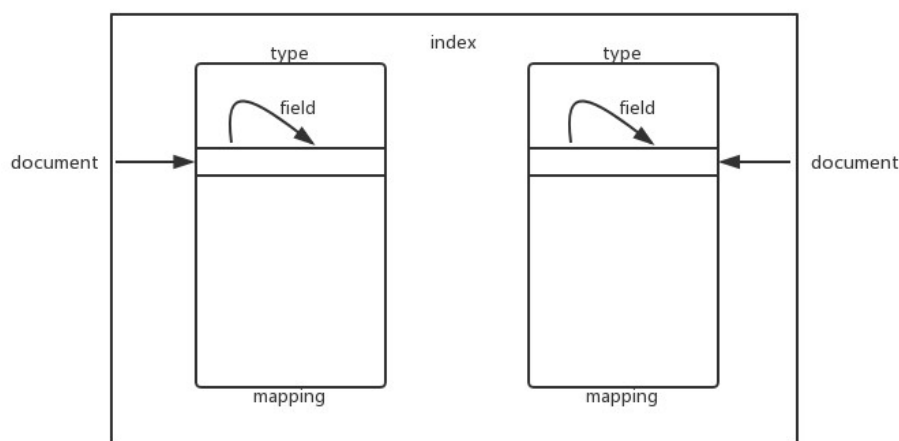


图 2-2 索引的工作原理

Figure 2-2 Operating Principle of Index

一个索引，可以拆分成多个 `shard`，每个 `shard` 存储部分数据。好处在于，一是支持横向扩展，比如数据量是 3T，3 个 `shard`，每个 `shard` 是 1T 的数据，若现在数据量增加到 4T，怎么扩展，很简单，重新建一个有 4 个 `shard` 的索引，将数据导进去；二是提高性能，数据分布在多个 `shard`，即多台服务器上，所有的操作，都会多台机器上并行分布式执行，提高了吞吐量和性能。

一个 `shard` 的数据有多个备份，即每个 `shard` 都有一个 `primary shard`，负责写入数据，但是还有几个 `replica shard`。当 `primary shard` 写入数据后，会将数据同步到其他几个 `replica shard` 上去。

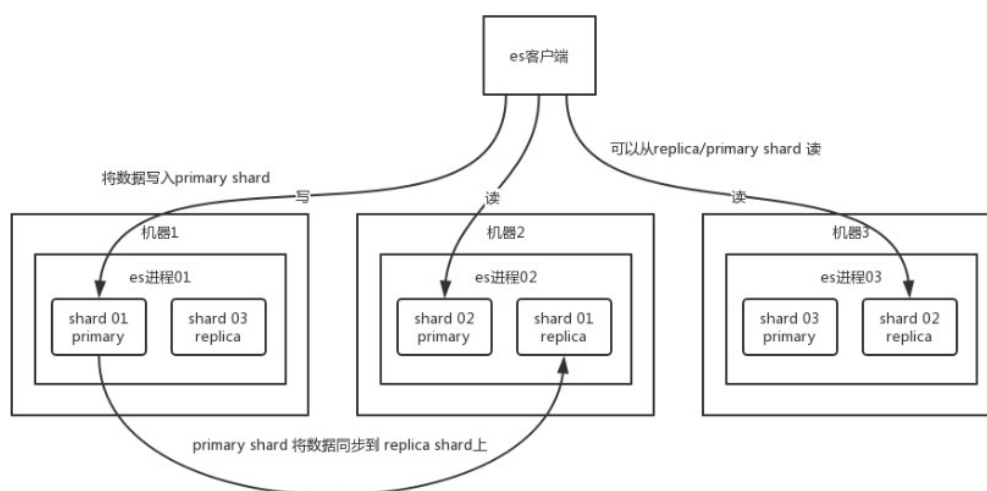


图 2-3 分片与副本的工作原理

Figure 2-3 Operating Principle of Shard and Replica

ES 集群多个节点，会自动选举一个节点为 **master** 节点，这个 **master** 节点就是干一些管理的工作，比如维护索引元数据、负责切换 **primary shard** 和 **replica shard** 身份等。要是 **master** 节点宕机了，会重新选举一个节点为 **master** 节点。

如果是非 **master** 节点宕机了，会由 **master** 节点，让那个宕机节点上的 **primary shard** 的身份转移到其他机器上的 **replica shard**。如果修复了那个宕机机器，重启之后，**master** 节点会控制将缺失的 **replica shard** 分配过去，同步后续修改的数据之类，让集群恢复正常。

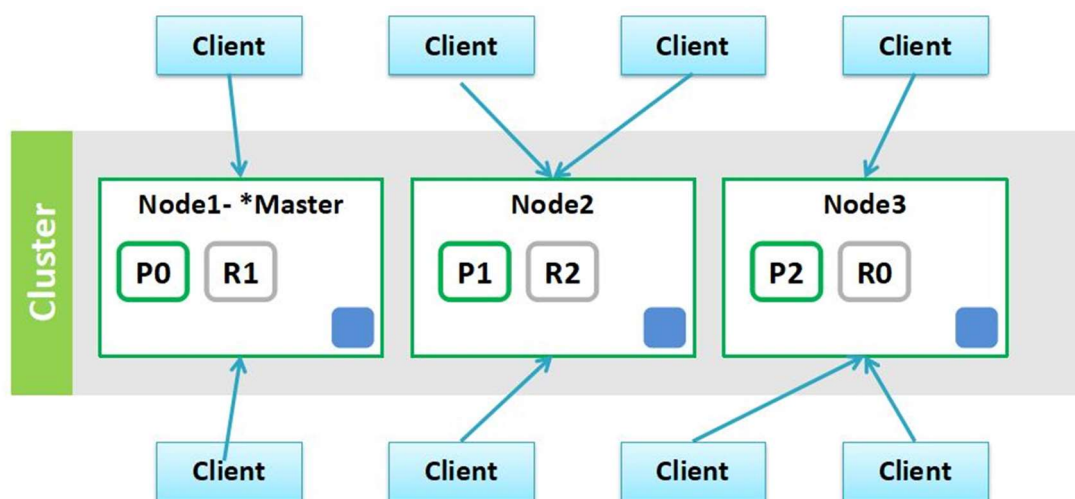


图 2-4 集群架构

Figure 2-4 Cluster Architecture

3 Elasticsearch 写入、查询的原理

3 Elasticsearch Writes and Searches Data Procedure

3.1 ES 写入数据原理 (ElasticSearch Writes Data Procedure)

客户端选择一个 node 发送请求, 这个 node 就是 coordinate node(协调节点)。coordinate node 对 document 进行路由, 将请求转发给对应的 node。实际的 node 上的 primary shard 处理请求, 然后将数据同步到 replica shard。coordinate node 如果发现 primary shard 和 replica shard 都写入成功后, 就返回响应结果给客户端。

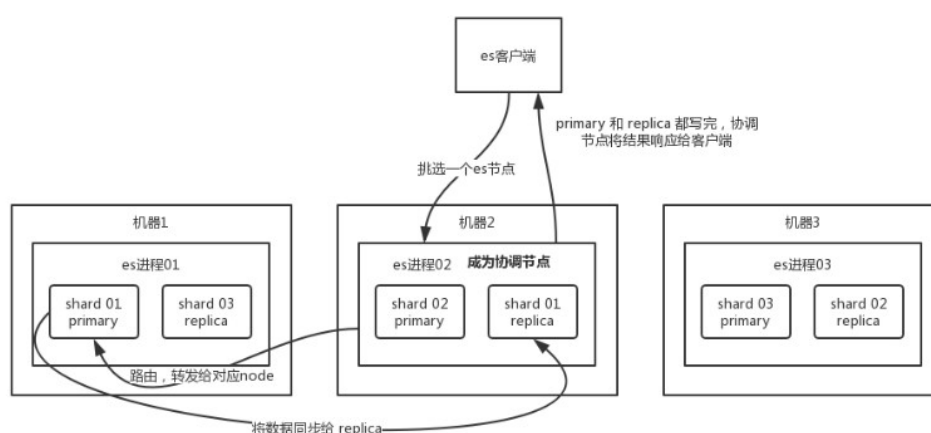


图 3-1 ES 写入数据过程

Figure 3-1 Elasticsearch Writes Data Process

数据先写入内存 buffer, 在 buffer 里的时候数据是搜索不到的, 同时将数据写入 translog 日志文件。

如果 buffer 快满了, 或者到一定时间, 就会将内存 buffer 数据 refresh 到一个新的 segment file 中, 但是此时数据不是直接进入 segment file 磁盘文件, 而是先进入 os cache。这个过程就是 refresh。

每隔 1 秒钟, ES 将 buffer 中的数据写入一个新的 segment file, 即每秒钟会产生一个新的磁盘文件 segment file, 这个 segment file 中就存储最近 1 秒内 buffer 中写入的数据。如果 buffer 里面此时没有数据, 则不会执行 refresh 操作, 如果 buffer 里有数据, 默认 1 秒钟执行一次 refresh 操作, 刷入一个新的 segment file 中。

操作系统里面, 磁盘文件都有一个东西, 叫做 os cache, 即操作系统缓存, 就是说数据写入磁盘文件之前, 会先进入 os cache, 就是操作系统级别的一个内存缓存中去。只要 buffer 中的数据被 refresh 操作刷入 os cache 中, 这个数据就可以被搜索到了, 该过程类似于其他分布式数据库的溢写过程。

ES 是准实时的，即 NRT，全称 near real-time。默认是每隔 1 秒 refresh 一次，所以 ES 是准实时的，因为写入的数据 1 秒之后才能被看到。可以通过 ES 的 RESTful API 或者 java API，手动执行一次 refresh 操作，就是手动将 buffer 中的数据刷入 os cache 中，让数据立马就可以被搜索到。只要数据被输入 os cache 中，buffer 就会被清空，因为不需要保留 buffer 了，数据在 translog 里面已经持久化到磁盘一份。

重复上面的步骤，新的数据不断进入 buffer 和 translog，不断将 buffer 数据写入一个又一个新的 segment file 中去，每次 refresh 后，将 buffer 清空，translog 保留。随着这个过程推进，translog 会变得越来越长。当 translog 达到一定长度的时候，就会触发 commit 操作。

commit 操作的第一步，就是将 buffer 中现有数据 refresh 到 os cache 中去，清空 buffer。然后，将一个 commit point 写入磁盘文件，里面标识着这个 commit point 对应的所有 segment file，同时强行将 os cache 中目前所有的数据都 fsync 到磁盘文件中去。最后清空现有 translog 日志文件，重启一个 translog，此时 commit 操作完成。这个 commit 操作叫做 flush，默认 30 分钟自动执行一次，但如果 translog 过大，也会触发 flush。flush 操作就对应着 commit 的全过程，可以通过 ES API，手动执行 flush 操作，手动将 os cache 中的数据 fsync 强刷到磁盘上去。

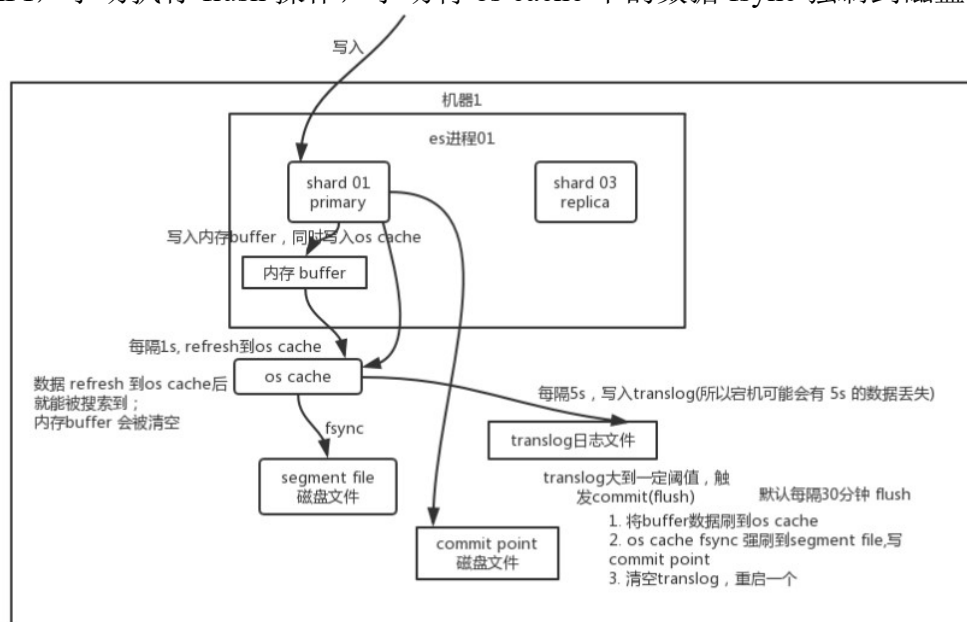


图 3-2 ES 写入数据原理

Figure 3-2 Elasticsearch Writes Data Procedure

3.2 ES 查询数据原理 (ElasticSearch Searches Data Procedure)

ES 擅长进行全文检索，根据关键词搜索，返回包含关键词的 document。

首先，客户端发送请求到一个 coordinate node。协调节点将搜索请求转发到所有的 shard 对应的 primary shard 和 replica shard。每个 shard 将自己的搜索结果返回给协调节点，由协调节点进行数据的合并、排序、分页等操作，产出最终结

果。接着由协调节点根据 doc id 去各个节点上拉取实际的 document 数据，最终返回给客户端。

ES 的内核是 Lucene，Lucene 的查询原理即是 ES 的查询原理。在 ES 中查询是基于 segment。每个 segment 可以看做是一个独立的 subindex，在建立索引的过程中，ES 会不断的 flush 内存中的数据持久化形成新的 segment。多个 segment 也会不断的被 merge 成一个大的 segment，在老的 segment 还有查询在读取的时候，不会被删除，没有被读取且被 merge 的 segment 会被删除。以表 3-1 为例，说明这个原理。

DocId	Name	Age	Id
1	Alice	18	101
2	Alice	20	102
3	Alice	21	103
4	Alan	21	104
5	Alan	18	105

表 3-1 学生表

Table 3-1 A Sample of Table

首先建立倒排索引，倒排索引是 ES 查询中非常关键的内容，在搜索引擎中，每个文档都有一个对应的文档 ID，文档内容被表示为一系列关键词的集合。例如，文档 1 经过分词，提取了 20 个关键词，每个关键词都会记录它在文档中出现的次数和出现位置。

那么，倒排索引就是关键词到文档 ID 的映射，每个关键词都对应着一系列文件，这些文件中都出现了该关键词。

如果为了查询学生的姓名，会建立基于 name 的倒排索引。

WordId	Word	DocIds
1	Alice	[1,2,3]
2	Alan	[4,5]

表 3-2 倒排索引

Table 3-2 Inverted Index

其中 Alice, Alan 是 term。倒排本质上就是基于 term 的反向列表，方便进行属性查找。到这里我们有个很自然的问题，如果 term 非常多，就需要在 ES 里面引入 term dictionary 的概念，也就是 term 的字典。term 字典里可以按照 term 进行排序，用一个二分查找就可以定为这个 term 所在的地址。这样的复杂度是 $\log N$ ，在 term 很多，内存放不下的时候，效率还是需要进一步提升。可以用一个 hashmap，当有一个 term 进入，hash 继续查找倒排链^[4]。

接下来，面对不同的使用场景，ES 会采取不同的数据结构存储来存储 term。

为了方便实现 `rangequery`（范围查询）或者前缀，后缀等复杂的查询语句，ES 使用了 FST 数据结构来存储 `term` 字典。

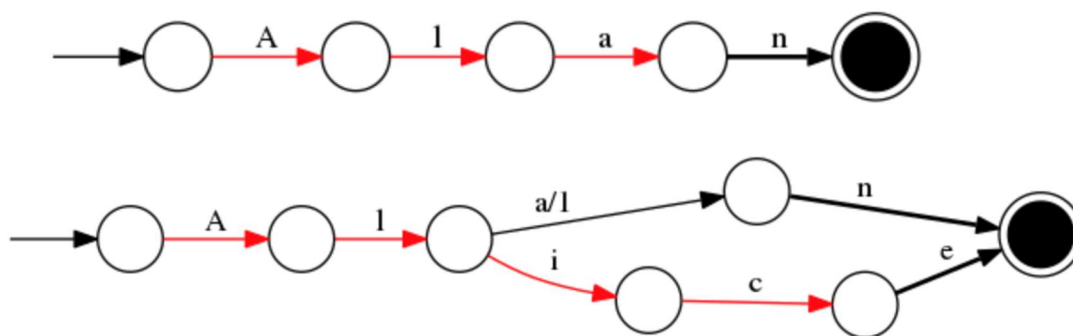


图 3-3 插入 Term

Figure 3-3 Insert the Terms

这样就得到了一个有向无环图，有这样一个数据结构，就可以很快查找某个学生姓名是否存在。FST 在 `term` 查询上可能相比 `hashmap` 并没有明显优势，甚至会慢一些。但是在范围查询，前缀搜索以及压缩率上都有明显的优势。

而为了能够快速查找 `Docid`，ES 会采用 `SkipList` 这一数据结构^[5]。

使用 `SkipList` 以后比如要查找 `docid=12`，原来需要一个个扫原始链表，1，2，3，5，7，8，10，12。有了 `SkipList` 先访问第一层发现大于 12，进入第 0 层到 3，8，发现 15 大于 12，然后进入原链表的 8 继续向下经过 10 和 12。



图 3-4 SkipList 查找

Figure 3-4 SkipList Lookup

存储完 `term`，接下来需要进行倒排合并。

假如查询条件是 `name="Alice"`，首先在 `term` 字典中定位是否存在这个 `term`，如果存在的话进入这个 `term` 的倒排索引，并根据参数设定返回分页返回结果即可。这类查询，在数据库中使用二级索引也是可以满足。但假如有多条件，例如需要按名字或者年龄单独查询，也需要进行组合 `name="Alice" and age=18` 的查询，那么使用传统二级索引方案，需要建立两张索引表，然后分别查询结果后进行合并，这样如果 `age=18` 的结果过多的话，查询合并会很耗时。

在 ES 中，假设有图 3-5 的几个 `term`。

首先在 `termA` 开始遍历，得到第一个元素 `docId=1`，Set `currentDocId=1`。在 `termB` 中 `search(currentDocId)=1`，因为 `currentDocId==1`，继续如果 `currentDocId` 和

返回的不相等，执行 2，然后继续到 termC 后依然符合，返回结果 `currentDocId = termC` 的 `nextItem` 然后继续步骤 3 依次循环。直到某个倒排链到末尾。

termA	1	3	4	5	8	9	
termB	1	6	9				
termC	1	2	5	8	9	10	11

图 3-5 倒排合并
Figure 3-5 Inverted Merger

这便是 ES 在通常情况下查询的原理。ES 的查询设计之初是为了进行文档类型的搜索，如果需要进行对数字的范围查找（一般指二维以上的坐标范围），将不使用常用的 FST-Inverted Merger 模式，转而使用 BKDTree-BST 模式，在此不做赘述。

4 Elasticsearch 与 HBase 的比较

4 Comparison of Elasticsearch and HBase

4.1 数据存储方式（ElasticSearch Writes Data Procedure）

HBase 则是以列为单位存储数据，每一列就是一个 key-value，HBase 的表列（属性）不用提前定义，而且列可以动态扩展，支持直接插入。

key	value
<1,name>	Alan
<1,age>	18
<1,id>	001

表 4-1 键值对存储

Table 4-1 Key-value Pair Storage

ES 比较灵活，索引中的 field 类型可以提前定义（定义 mapping），也可以不定义，如果不定义，会有一个默认类型，不过出于可控性考虑，关键字段最好提前定义好。

DocId	Doc
1	谷歌地图之父跳槽 Facebook
2	谷歌地图之父加盟 Facebook
3	谷歌地图创始人拉斯离开谷歌加盟腾讯
4	谷歌地图之父跳槽 Facebook 与 Wave 项目
5	谷歌地图之父拉斯加盟社交网站 Facebook

表 4-2 索引存储

Table 4-2 Index Storage

4.2 容灾方式（Disaster Recovery Mode）

HBase 分为两层，一层为 NoSql service，一层是分布式文件系统 HDFS。HBase 中 region 分布在不同的 regionserver 上，client 端通过 meta 表来定位数据在哪个 regionserver 的 region 上，然后获取数据，但是数据有可能并不一定在该 regionserver 本地保存，每个 region 都知道自己对应的数据在 HDFS 的哪些数据块上，最后通过访问 HDFS 来获取数据，尤其当 HBase 和 HDFS 部署在不同的集群上时，数据的读写完全是通过 RPC 来实现，为了减少 RPC 的开销，保证服务稳定，往往会将 HBase 和 HDFS 部署在同一个集群。同理，当一个 regionserver 宕机，region 可以快速切换到别的 regionserver 上。

ES 的容灾也是采用写 log 的方式，与 HBase 不同的是，ES 的节点保存各自

的 log，这点跟 MySQL 类似，log 是存放在本地的，这也就存在和 MySQL 一样的问题，假如机器宕机或硬盘故障，log 数据也会丢失，所以 index 每个 shard 也有主备，默认配置是一个 primary shard，一个 replica shard，当然也可以配置多个 replica。默认情况下，primary shard 首先接收 client 端发送过来的数据，然后将数据同步到 replica shard 中，当 replica shard 也写入成功后，才会告知 client 数据已正确写入，这样就防止数据还没写入 replica shard 时，primary 掉线导致的数据丢失。

4.3 查询方式（Search Mode）

HBase 不支持二级索引，它只有一个主键索引，采用 LSM 树。HBase 的查询实现只提供两种方式：按指定 RowKey 获取唯一一条记录，get 方法。按指定的条件获取一批记录，scan 方法。实现条件查询功能使用的就是 scan 方式，scan 可以通过 setFilter 方法添加过滤器，这也是分页、多条件查询的基础^[6]。

ES 的查询最主要的优势在于它基于倒排索引构建的全文检索能力，这是 Hbase 所不具备的能力。ES 除了倒排索引，也有基于 DocValues 的正排索引，DocValues 采用列式存储，可以比较快速地实现聚合和排序查询场景。通过 ES 提供的 DSL 和 Aggregations，以及 SQL 查询引擎，用户可以方便的实现各式各样的查询需求。

4.4 优缺点总结（Summary of Advantages and Disadvantages）

ElasticSearch 既是搜索引擎也是数据库，支持全中文搜索。可以自动建立索引，实现高性能的聚合查询，面对复杂查询也有不错的性能。还提供统计功能，并且提供的 RESTful 接口非常好用，配上 Kibana 还可以进行图形化展示，第三方插件也很丰富。日志能够每隔一段时间归一次档，创建新的 index，做到冷热分离。

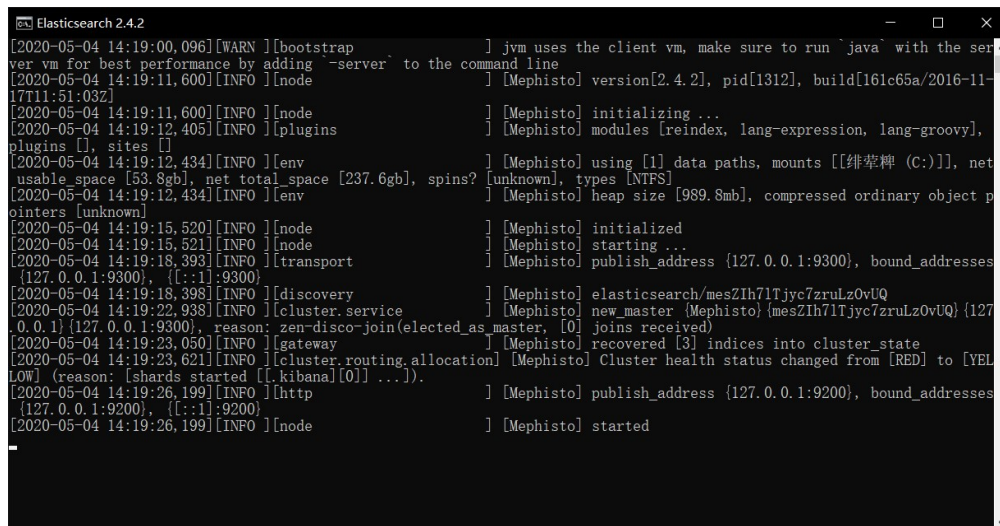
ElasticSearch 最明显的就是字段类型无法修改，只有文档存储类型。写入性能较低。高性能资源消耗，需要占用堆内存。

HBase 继承了 Hadoop 项目的优点，适合对海量数据的支持，有极强的横向扩展能力。使用廉价的 PC 机就能够搭建起海量数据处理的大数据集群。

HBase 对数据的读取带来了局限，只有同一列族的数据才能够放在一起，而且所有的查询都必须依赖于 key，这就使得很多复杂的查询难以实现。同时 HBase 是很重的产品，依赖于很多 Hadoop 组件。

5 Elasticsearch 的一个简单实例

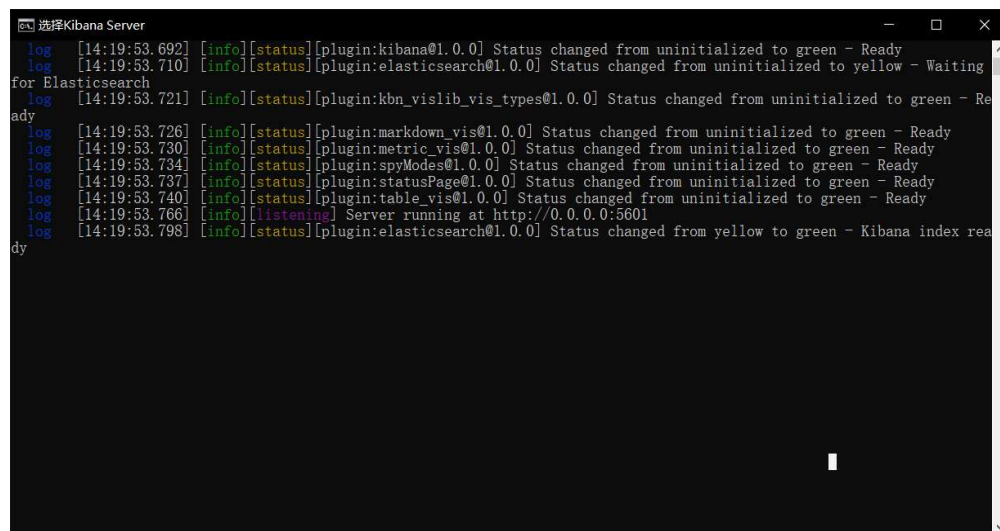
本网站为一个线上商城项目，源代码见附录，使用 JPA 存储层技术与 Elasticsearch 分布式数据库技术，分别启动 ES、Kibana、IDEA 控制台，运行 Springboot 内置 Tomcat 访问网站，进行同样的搜索数次，在控制台查看结果。



```
Elasticsearch 2.4.2
[2020-05-04 14:19:00,096][WARN ][bootstrap] jvm uses the client vm, make sure to run `java` with the server
vm for best performance by adding `-server` to the command line
[2020-05-04 14:19:11,600][INFO ][node] [Mephisto] version[2.4.2], pid[1312], build[161c65a/2016-11-17T11:51:03Z]
[2020-05-04 14:19:11,600][INFO ][node] [Mephisto] initializing ...
[2020-05-04 14:19:12,405][INFO ][plugins] [Mephisto] modules [reindex, lang-expression, lang-groovy],
plugins [], sites []
[2020-05-04 14:19:12,434][INFO ][env] [Mephisto] using [1] data paths, mounts [[/ (fat32) (C:)], net
usable space [53.8gb], net total_space [237.6gb], spins? [unknown], types [NTFS]
[2020-05-04 14:19:12,434][INFO ][env] [Mephisto] heap size [989.8mb], compressed ordinary object p
ointers [unknown]
[2020-05-04 14:19:15,520][INFO ][node] [Mephisto] initialized
[2020-05-04 14:19:15,521][INFO ][node] [Mephisto] starting ...
[2020-05-04 14:19:18,393][INFO ][transport] [Mephisto] publish_address [127.0.0.1:9300], bound_addresses
{127.0.0.1:9300}, {[::1]:9300}
[2020-05-04 14:19:18,398][INFO ][discovery] [Mephisto] elasticsearch/mesZiH71Tjyc7zruLzOvUQ
[2020-05-04 14:19:22,938][INFO ][cluster.service] [Mephisto] new_master {Mephisto}{mesZiH71Tjyc7zruLzOvUQ}{127
.0.0.1}{127.0.0.1:9300}, reason: zen-disco-join(elected_as_master, [0] joins received)
[2020-05-04 14:19:23,050][INFO ][gateway] [Mephisto] recovered [3] indices into cluster_state
[2020-05-04 14:19:23,621][INFO ][cluster.routing.allocation] [Mephisto] Cluster health status changed from [RED] to [YEL
LOW] (reason: [shards started [[.kibana][0]] ...)).
[2020-05-04 14:19:26,199][INFO ][http] [Mephisto] publish_address [127.0.0.1:9200], bound_addresses
{127.0.0.1:9200}, {[::1]:9200}
[2020-05-04 14:19:26,199][INFO ][node] [Mephisto] started
```

图 5-1 启动 Elasticsearch

Figure 5-1 Start the Elasticsearch



```
选择Kibana Server
log [14:19:53.692] [info][status][plugin:kibana@1.0.0] Status changed from uninitialized to green - Ready
log [14:19:53.710] [info][status][plugin:elasticsearch@1.0.0] Status changed from uninitialized to yellow - Waiting
for Elasticsearch
log [14:19:53.721] [info][status][plugin:kbn_vislib_vis_types@1.0.0] Status changed from uninitialized to green - Re
ady
log [14:19:53.726] [info][status][plugin:markdown_vis@1.0.0] Status changed from uninitialized to green - Ready
log [14:19:53.730] [info][status][plugin:metric_vis@1.0.0] Status changed from uninitialized to green - Ready
log [14:19:53.734] [info][status][plugin:spyModes@1.0.0] Status changed from uninitialized to green - Ready
log [14:19:53.737] [info][status][plugin:statusPage@1.0.0] Status changed from uninitialized to green - Ready
log [14:19:53.740] [info][status][plugin:table_vis@1.0.0] Status changed from uninitialized to green - Ready
log [14:19:53.766] [info][listening] Server running at http://0.0.0.0:5601
log [14:19:53.798] [info][status][plugin:elasticsearch@1.0.0] Status changed from yellow to green - Kibana index rea
dy
```

图 5-2 启动 Kibana

Figure 5-2 Start the Kibana

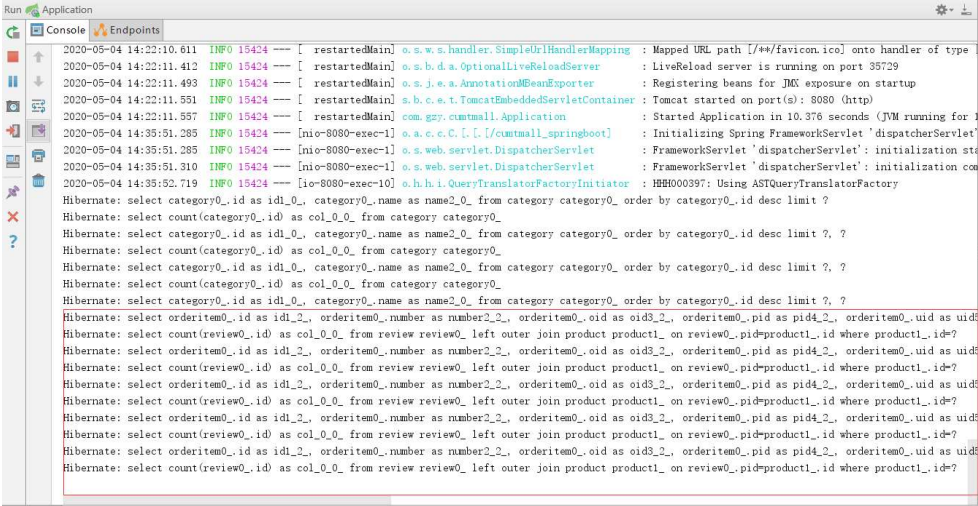


Figure 5-3 Startup Console

观察控制台输出，发现搜索同一个词条时，第一次会在控制台输出 SQL 语句，之后则不会。

该实例是本人在 JAVA Web 开发实验课所完成项目的一部分，通过调用 RESTful API 调用 ES，每一次的搜索优先查询 ES 中存储的内容，如果 ES 中查无此内容，则在 MySQL 中进行查询。

当多次搜索同一关键字时，在堆内存或者服务器的 **swarp** 分区中建立索引。之后每一次的搜索，ES 可在内存中完成高速查询，解决了搜索热点问题。

参考文献

- [1] Shay Banon. ElasticSearch 权威指南[EB/OL]. <https://www.elastic.co/guide>
- [2] Gormley C, Tong Z. Elasticsearch: the definitive guide: a distributed real-time search and analytics engine[M]. "O'Reilly Media, Inc.", 2015.
- [3] 周恒敏. 基于 ELASTICSEARCH 在企业大数据中的应用[D]. 对外经济贸易大学, 2019.
- [4] Culpepper J S, Moffat A. Efficient set intersection for inverted indexing[J]. ACM Transactions on Information Systems (TOIS), 2010, 29(1): 1-25.
- [5] Shavit N, Lotan I. Skiplist-based concurrent priority queues[C]//Proceedings 14th International Parallel and Distributed Processing Symposium. IPDPS 2000. IEEE, 2000: 263-268.
- [6] Taylor R C. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics[C]//BMC bioinformatics. BioMed Central, 2010, 11(S12): S1.
- [7] Paro A. ElasticSearch cookbook[M]. Packt Publishing Ltd, 2015.
- [8] 白俊, 郭贺彬. 基于 ElasticSearch 的大日志实时搜索的软件集成方案研究[J]. 吉林师范大学学报: 自然科学版, 2014, 35(1): 85-87.
- [9] 李传根. Elasticsearch 数据存储策略研究[D]. 重庆邮电大学, 2019.
- [10] Schreter I, Gottipati C, Legler T. Inverted indexing: U.S. Patent 9,971,770[P]. 2018-5-15.
- [11] Özsu M T, Valduriez P. Principles of distributed database systems[M]. Englewood Cliffs: Prentice Hall, 1999.
- [12] Corbett J C, Dean J, Epstein M, et al. Spanner: Google's globally distributed database[J]. ACM Transactions on Computer Systems (TOCS), 2013, 31(3): 1-22.
- [13] 林子雨, 赖永炫, 林琛, 等. 云数据库研究[D]. 中国科学院软件研究所| 中国计算机学会, 2012.

附录

第 1 部分 线上商城网站

http://39.100.31.120/cumtmall_springboot/home

第 2 部分 部分源码

ES 的配置，ES 和 JPA 分别指定不同的包名，因为 JPA 的 DAO 做了 Redis 的连接。

```
@EnableElasticsearchRepositories(basePackages = "com.gzy.cumtmall.es")
@EnableJpaRepositories(basePackages = {"com.gzy.cumtmall.dao", "com.gzy.cumtmall.pojo"})
public class Application {
    static {
        PortUtil.checkPort(6379, "Redis 服务端", true);
        PortUtil.checkPort(9300, "ElasticSearch 服务端", true);
        PortUtil.checkPort(5601, "Kibana 工具", true);
    }
}
```

增加，删除，修改的时候，除了通过 ProductDAO 对数据库产生影响之外，还要通过 ProductESDAO 同步到 ES。

```
public void add(Product bean) {
    productDAO.save(bean);
    productESDAO.save(bean);
}

public void delete(int id) {
    productDAO.delete(id);
    productESDAO.delete(id);
}

public void update(Product bean) {
    productDAO.save(bean);
    productESDAO.save(bean);
}
```

初始化数据到 ES，所以刚开始查询，先检查 ES 有没有数据，如果没有，就把数据同步到 ES 中。

```
private void initDatabase2ES() {
    Pageable pageable = new PageRequest(0, 5);
    Page<Product> page = productESDAO.findAll(pageable);
    if (page.getContent().isEmpty()) {
        List<Product> products = productDAO.findAll();
        for (Product product : products) {
            productESDAO.save(product);
        }
    }
}
```

通过 ProductESDAO 到 Elasticsearch 中进行查询。

```
public List<Product> search(String keyword, int start, int size) {
    initDatabase2ES();
    FunctionScoreQueryBuilder functionScoreQueryBuilder =
QueryBuilders.functionScoreQuery()
        .add(QueryBuilders.matchPhraseQuery("name", keyword),
            ScoreFunctionBuilders.weightFactorFunction(100))
        .scoreMode("sum")
        .setMinScore(10);
    Sort sort = new Sort(Sort.Direction.DESC, "id");
    Pageable pageable = new PageRequest(start, size, sort);
    SearchQuery searchQuery = new NativeSearchQueryBuilder()
        .withPageable(pageable)
        .withQuery(functionScoreQueryBuilder).build();
    Page<Product> page = productESDAO.search(searchQuery);
    return page.getContent();
}
```