ElasticSearch

倒排索引

核心思想是：从词找到文档，而不是从文档找到词

单词 映射到 包含该单词的文档 **ID** 列表，从而加速全文搜索。相比传统的 正向索引（逐个扫描文档），倒排索引可以 快速定位包含某个词的所有文档，大幅提升查询效率。

> 结构？

倒排索引主要由 词项（**Term**） 和 倒排列表（**Posting List**） 组成：

词项（**Term**）：索引中的每个唯一单词。

倒排列表（**Posting List**）：记录包含该单词的 文档 **ID** 及其 出现次数、位置信息（可选）。

> 用处？对比*sql?* 为什么快？

适合全文搜索（模糊匹配）

底层数据结构是倒排列表

主要包含 倒排表（**Posting List**） 和 跳表（**Skip List**）等优化机制。

分词

分析器=分词器+过滤器

分词后的文本 可以更灵活地匹配查询，提高搜索命中率。

直接索引整句 `"机器学习很重要"`，无法匹配 `"学习"` 或 `"机器"`，影响搜索体验。

中文需要分词器，否则拆分后无意义。（词和字的区分）

> 怎么分？

ik分词器

IK_SMART	粗粒度分词	**["机器学习", "重要"]**

IK_MAX_WORD	细粒度分词（尽可能多拆分）	**["机器", "学习", "机器学习", "很", "重要"]**

> *ik*分词器

IK 分词器内置了一个 大规模中文词库，加载后可以用来匹配文本。

例如，输入 "中国科学院"，IK 会检查 "中国"、"科学院"、"中国科学院" 是否在词典中。

> 过滤器

过滤器 会对生成的词元进行进一步的处理，如小写化、去除停用词、词干提取、同义词替换等。

搜索 查询过程

> 查询解析

解析查询：识别查询类型（全文搜索、精准匹配、聚合等）。

分词（如果是全文搜索）：对搜索关键词进行分词处理，以匹配索引中的倒排索引词项。

> 查询分发

- Elasticsearch 识别 索引在哪些分片（**Shard**）上。
- 主节点（**Coordinator Node**） 负责把查询发送到所有相关的分片（主分片或副本分片）。

> 分片搜索

每个分片 运行搜索，基于 倒排索引 进行匹配。

评分（**Scoring**）：计算文档的 **TF-IDF/BM25** 相关性分数。

Top-N 结果：每个分片本地返回排名靠前的文档（如前 10 条）。

> 聚合结果

协调节点收集各分片返回的候选文档。

统一排序（全局 Top-N），确保最相关的结果靠前。

(**BM25** 相关性计算，各返回 **Top-10** 结果)

> 什么是*BM25* 相关性计算

BM25（Best Matching 25）是 一种基于概率模型的文档排序算法，用于计算文档与查询之间的相关性分数。它是 Elasticsearch 默认的 相关性评分算法，广泛用于全文搜索引擎中。

BM25 通过 词频（**TF**） 和 文档长度（**Document Length**） 来评估文档与查询的匹配度

副本

拷贝多少份

通过将主分片和副本分片分布在不同的节点（服务器）上，确保即使一个节点失败，副本仍然可用，数据不会丢失，确保了系统的可用性。

我的项目里是三主三副　三个主分片 每个主分片一个副分片

三个服务器节点 每个服务器存一个主分片 和另外一个追分片的副分片

用`_cat/shards?v`来查分片对应节点号

分片

索引数据被分割成多个 分片（**Shard**），每个分片都可以分布在集群中的不同节点上。每个索引都有一个或多个分片，而 分片的具体位置 是通过 **Elasticsearch** 的路由算法 来确定的。

对文档id进行哈希

如果没有指定路由键，Elasticsearch 会查询该索引的 所有分片。每个分片都会并行搜索，最终将结果合并。

有的话就去某一个分片查

索引** 在 Elasticsearch 中是会被 分片（Sharded）的，每个索引都会根据设定的主分片数量被分成多个 分片（Shard）。在进行查询时，Elasticsearch 会自动在 所有分片 上进行搜索操作，然后整合结果返回给用户。

其他八股

是一个基于 **Lucene** 的全文搜索引擎，数据存储在 倒排索引 中，适合快速搜索和大规模数据分析。

不支持传统的事务（ACID），主要关注快速检索和分析，适合批量写入和实时搜索。

> 插入的过程？

1、spring后台 post和 **PUT** 请求将数据插入 Elasticsearch。

2、插入json（字段这些，关联一个主键id)

3、Elasticsearch 收到插入请求时，它会根据 路由算法（通常基于文档 ID）确定将数据插入到哪个 主分片。每个文档会通过哈希算法，结合 文档 **ID** 或者 指定的路由字段 来计算出它应该存储在哪个分片。

4、插入操作：插入的数据会存储到分片的内存缓冲区中。此时，数据不会立即写入磁盘，而是先存储在内存中以提高写入速度。

刷新（**Refresh**）：数据存储在内存中的缓冲区中，直到 Elasticsearch 进行“刷新”操作，将内存中的数据写入磁盘的实际存储文件中。这通常会在一定的时间间隔后自动触发，或者可以手动触发刷新操作。

5、插入的数据会首先写入到目标 主分片。如果该索引配置了副本分片，Elasticsearch 会将主分片的数据复制到相应的副本分片中。

6、 在默认情况下，数据会在写入后立即可查询，前提是 Elasticsearch 已经执行了刷新操作（通常在 1 秒钟左右）。

7、每个文档存入 Elasticsearch 后，它会被 分词，即文档中的文本内容会经过分词器的处理，将字符串拆分成一系列的词项（tokens），然后这些词项会存储到倒排索引中。

> 结合项目？

有新的申请表后 返回一个主键id 放入mq 里面消费 消费者反查Mysql 组装需要的数据 再插入到mysql里面

> 为什么要用到*es?*

审核后台需要按照条件筛选申请表 还有根据申请单名称模糊查询，但是每次走mysql筛选会很慢 所以要用到ES4

> 具体字段？

申请单名称

申请单状态

是否有图片

申请单类型

申请单工单id

身份证号

创建时间

修改时间

> 文本相似度

词越多 越匹配

> 深翻页问题？

记住 查询是所有分片查　会跟mysql超大分页问题一样　es有匹配度机制 把对应的分页结果返回 每次翻页都会查一次es　解决措施：放redis

重提：mysql解决超大分页问题是靠覆盖索引(先用联合索引查主键，再返回结果)

> 脑裂问题？

和redis一样

> 优化？

可以考虑冷热数据

> 缺点？

ES 适合全文搜索，但不擅长复杂 SQL 查询

Elasticsearch 不是强一致性的，它采用 最终一致性，

内存占用高，容易 OOM

ES 需要手动调优，否则容易出现：

- 分片分配不均，导致某些节点压力过大。
- 查询和写入负载不均衡，影响整体性能。
- 节点崩溃时，数据恢复较慢，可能影响可用性