Elasticsearch 索引设计实战指南

Part 2

4、中文分词

中文检索常用ES IK分词插件

IK分词器的两种分词模式

* ik\_max\_word: 会将文本做最细粒度的拆分，比如会将"中华人民共和国国歌"拆分为"中华人民共和国,中华人民,中华,华人,人民共和国,人民,人,民,共和国,共和,和,国国,国歌"，会穷尽各种可能的组合；
* ik\_smart: 会做最粗粒度的拆分，比如会将"中华人民共和国国歌"拆分为"中华人民共和国,国歌"。

4.1 坑 1：分词选型

实际业务中：建议适用ik\_max\_word分词 + match\_phrase短语检索。

原因：ik\_smart有覆盖不全的情况，数据量大了以后，即便 reindex 能满足要求，但面对极大的索引的情况，reindex 的耗时我们承担不起。建议ik\_max\_word一步到位。

4.2 坑 2：ik 要装集群的所有机器吗？

建议：安装在集群的所有节点上。

4.3 坑 3：ik 匹配不到怎么办？

* 方案1：扩充 ik 开源自带的词库+动态更新词库；原生的词库分词数量级很小，基础词库尽量更大更全，网上搜索一下“搜狗词库“。

动态更新词库：可以结合 [mysql](https://cloud.tencent.com/product/cdb?from=10680)+ik 自带的更新词库的方式动态更新词库。

更新词库仅对新创建的索引生效，部分老数据索引建议使用 reindex 升级处理。

* 方案2：采用字词混合索引的方式，避免“明明存在，但是检索不到的”场景。

5、检索类型选型

* text 类型作用：分词，将大段的文字根据分词器切分成独立的词或者词组，以便全文检索。

适用于：email 内容、某产品的描述等需要分词全文检索的字段；

不适用：排序或聚合（Significant Terms 聚合例外）

* keyword 类型：无需分词、整段完整精确匹配。

适用于：email 地址、住址、状态码、分类 tags。

例

PUT zz\_test #创建索引

{

"mappings": {

"doc": {

"properties": {

"title": {

"type": “text", #该属性的基础类型

“analyzer":"ik\_max\_word", #该属性的基础分析器

"fields": {

"kw": { # 该属性的Multi-fields with multiple analyzers（多分析器处理多字段）用子属性名 本文的下文会多次用到

"type": "keyword",

"ignore\_above": 256

}

}

}

}

}

}

}

GET zz\_test/\_mapping

PUT zz\_test/doc/1 #写入文档

{

"title":"锤子加湿器官方致歉,难产后临时推迟一个月发货遭diss耍流氓"

}

POST zz\_test/\_analyze #查询分词详细

{

"text": "锤子加湿器官方致歉,难产后临时推迟一个月发货遭diss耍流氓",

"analyzer": "ik\_max\_word"

}

ik\_max\_word的分词结果如下：

锤子、锤、子、加湿器、湿、器官、官方、方、致歉、致、歉、难产、产后、后、临时、临、时、推迟、迟、一个、 一个、 一、个月、 个、 月、 发货、发、货、遭、diss、耍流氓、耍、流氓、氓。

5.1 term 精确匹配

* 核心功能：不受到分词器的影响，属于完整的精确匹配。
* 应用场景：精确、精准匹配。
* 适用类型：keyword。
* 举例：term 最适合匹配的类型是 keyword，如下所示的精确完整匹配：

POST zz\_test/\_search

{

"query": {

"term": {

"title.kw": "锤子加湿器官方致歉,难产后临时推迟一个月发货遭diss耍流氓"

}

}

}

* 注意：如下是匹配不到结果的。

POST zz\_test/\_search

{

"query": {

"term": {

"title": "锤子加湿器"

}

}

} #原因：1.对于 title 中的锤子加湿器，term 不会做分词拆分匹配的。2. ik\_max\_word 分词也是没有“锤子加湿器”这组关键词的。

5.2 prefix 前缀匹配

* 核心功能：前缀匹配。
* 应用场景：前缀自动补全的业务场景。
* 适用类型：keyword。

如下能匹配到文档 id 为 1 的文章。

POST zz\_test/\_search

{

"query": {

"prefix": {

"title.kw": "锤子加湿器"

}

}

}

5.3 wildcard 模糊匹配

* 核心功能：匹配具有匹配通配符表达式 keyword 类型的文档。支持的通配符：\*，它匹配任何字符序列（包括空字符序列）；？，它匹配任何单个字符。
* 应用场景：请注意，选型务必要慎重！此查询可能很慢多组关键词的情况下可能会导致宕机，因为它需要遍历多个术语。为了防止非常慢的通配符查询，通配符不能以任何一个通配符\*或？开头。
* 适用类型：keyword。

如下匹配，类似 MySQL 中的通配符匹配，能匹配所有包含加湿器的文章。

POST zz\_test/\_search

{

"query": {

"wildcard": {

"title.kw": "\*加湿器\*"

}

}

}

5.4 match 分词匹配

* 核心功能：全文检索，分词词项匹配。
* 应用场景：实际业务中较少使用，原因：匹配范围太宽泛，不够准确。
* 适用类型：text。
* 如下示例，title 包含"锤子"和“加湿器”的都会被检索到。

POST zz\_test/\_search

{

"profile": true, #设置顶级profile参数为true，开启搜索概要描述【开发用】

"query": {

"match": {

"title": "锤子加湿器"

}

}

}

5.5 match\_phrase 短语匹配

* 核心功能：match\_phrase 查询首先将查询字符串解析成一个词项列表，然后对这些词项进行搜索; 只保留那些包含 全部 搜索词项，且 位置"position" 与搜索词项相同的文档。
* 应用场景：业务开发中 90%+ 的全文检索都会使用 match\_phrase 或者 query\_string 类型，而不是 match。
* 适用类型：text。
* 注意：如下的检索是匹配不到结果的。

POST zz\_test/\_search

{

"query": {

"match\_phrase": {

"title": "锤子加湿器"

}

}

}

原因

POST zz\_test/\_analyze #对搜索关键字做分词分析

{

"text": "锤子加湿器",

"analyzer": "ik\_max\_word"

}

可得分词结果为

锤子， 锤，子， 加湿器， 湿，器

而：id为1的文档的分词结果：锤子, 锤, 子, 加湿器, 湿, 器官, …。

解决方案：字词组合索引

5.6 multi\_match 多组匹配

* 核心功能：match query 针对多字段的升级版本。
* 应用场景：多字段检索。
* 适用类型：text。
* 举例：

POST zz\_test/\_search

{

"query": {

"multi\_match": {

"query": "加湿器",

"fields": [

"title",

"content"

]

}

}

}

5.7 query\_string 类型

* 核心功能：支持与或非表达式+其他N多配置参数。
* 应用场景：业务系统需要支持自定义表达式检索。
* 适用类型：text。

POST zz\_test/\_search

{

"query": {

"query\_string": {

"default\_field": "title",

"query": "(锤子 AND 加湿器) OR (官方 AND 道歉)"

}

}

}

5.8 bool 组合匹配

* 核心功能：多条件组合综合查询。
* 应用场景：支持多条件组合查询的场景。
* 适用类型：text 或者 keyword。一个 bool 过滤器由三部分组成：

{

"bool" : {

"must" : [],

"should" : [],

"must\_not" : [],

"filter": []

}

}

* must ——所有的语句都 必须（must） 匹配，与 AND 等价。
* must\_not ——所有的语句都 不能（must not） 匹配，与 NOT 等价。
* should ——至少有一个语句要匹配，与 OR 等价。
* filter——必须匹配，运行在非评分&过滤模式。

检索类型选型小结

6、多表关联如何设计？

方案一：多表关联视图，视图同步 ES

* 步骤 1：提前关联好数据，将关联的表建立好视图，一个索引对应你的一个视图，并确认视图中数据的正确性。
* 步骤 2：ES 中针对每个视图定义好索引名称及 Mapping。
* 步骤 3：以视图为单位通过 logstash\_input\_jdbc 同步到 ES 中。

方案二：1 对 1 同步 ES

使用 ES 多索引建立相同的别名，针对别名检索到对应 ID 后再回 MySQL 通过关联 ID join 出需要的数据。

方案三：使用 Nested 做好关联

适用场景：1 对少量的场景。

举例：有一个文档描述了一个帖子和一个包含帖子上所有评论的内部对象评论。可以借助 Nested 实现。

方案四：使用 ES6.X+ 父子关系 Join 做关联

适用场景：1 对多量的场景。

举例：1 个产品和供应商之间是1对N的关联关系。

方案三、方案四选型对比：

7、实战中遇到过的坑

如果能重来，我会如何设计 Elasticsearch 系统？

来自累计近千万实战项目设计的思考。

* 坑1: 数据清洗一定发生在写入 es 之前！而不是请求数据后处理，那势必会降低请求速度和效率。
* 坑2：高亮功能不要重复造轮子，用原生就可以。
* 坑3：让 es 做他擅长的事，检索+不复杂的聚合，否则数据量+复杂的业务逻辑大会有性能问题。
* 坑4：设计的工作必须不要省！快了就是慢了，否则无休止的因设计缺陷引发的 bug 会增加团队的挫败感！
* 坑5：在给定时间的前提下，永远不会有完美的设计，必须相对合理的设计+重构结合，才会有相对靠谱的系统。
* 坑6：SSD 能提升性能，但如果系统业务逻辑非常复杂，换了 SSD 未必达到预期。
* 坑7：由于 Elasticsearch 不支持事务 ACID 特性，数据库作为实时数据补充，对于实时数据要求严格的场景，必须同时采取双写或者同步的方式。这样，一旦实时数据出现不一致，可以通过数据库进行同步递增更新。

次回

探究 | 明明存在，怎么搜索不出来呢？

<https://mp.weixin.qq.com/s?__biz=MzI2NDY1MTA3OQ==&mid=2247484287&idx=1&sn=e1f4b24f61d37d556828bbcd211707ac&chksm=eaa82b57dddfa241570730dde38b74a3ff9c36927fd84513b0136e4dc3ee7af18154290a27b2&scene=21#wechat_redirect>

参考

<https://blog.csdn.net/xsdxs/article/details/72853288>

<https://cloud.tencent.com/developer/article/1587375>

<https://mp.weixin.qq.com/s?__biz=MzI2NDY1MTA3OQ==&mid=2247484287&idx=1&sn=e1f4b24f61d37d556828bbcd211707ac&chksm=eaa82b57dddfa241570730dde38b74a3ff9c36927fd84513b0136e4dc3ee7af18154290a27b2&scene=21#wechat_redirect>