Doris 3.1 新版本解读（二）
半结构化能力全面升级之
倒排索引

姜凯 - Apache Doris Committer、飞轮科技资深技术专家

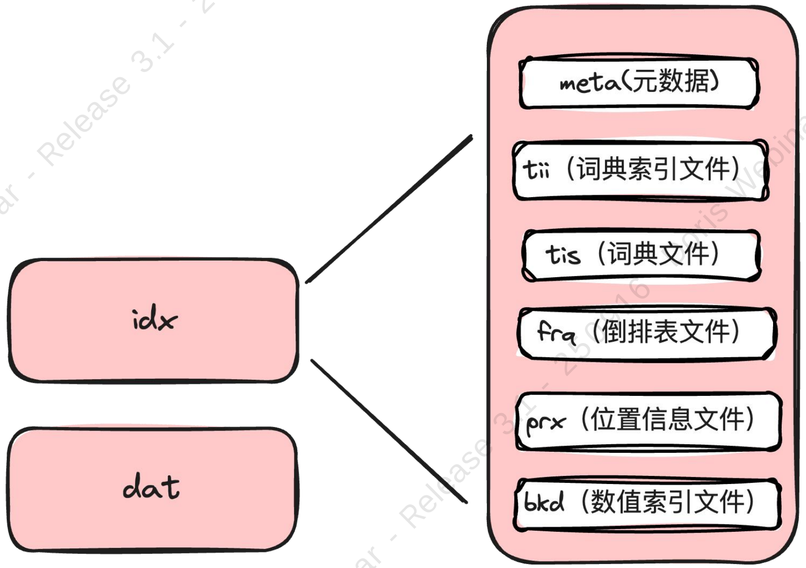
目录

倒排索引存储格式V3

新增三种内置分词器

自定义分词能力

Doris倒排索引文件结构简介



特点

参考Lucene

- Lucene 经典倒排索引结构
- Doris 借鉴其结构，但去掉了 Lucene 的 norms/forward 存储，以减轻存储和查询负担，更符合实时分析场景。

外挂式

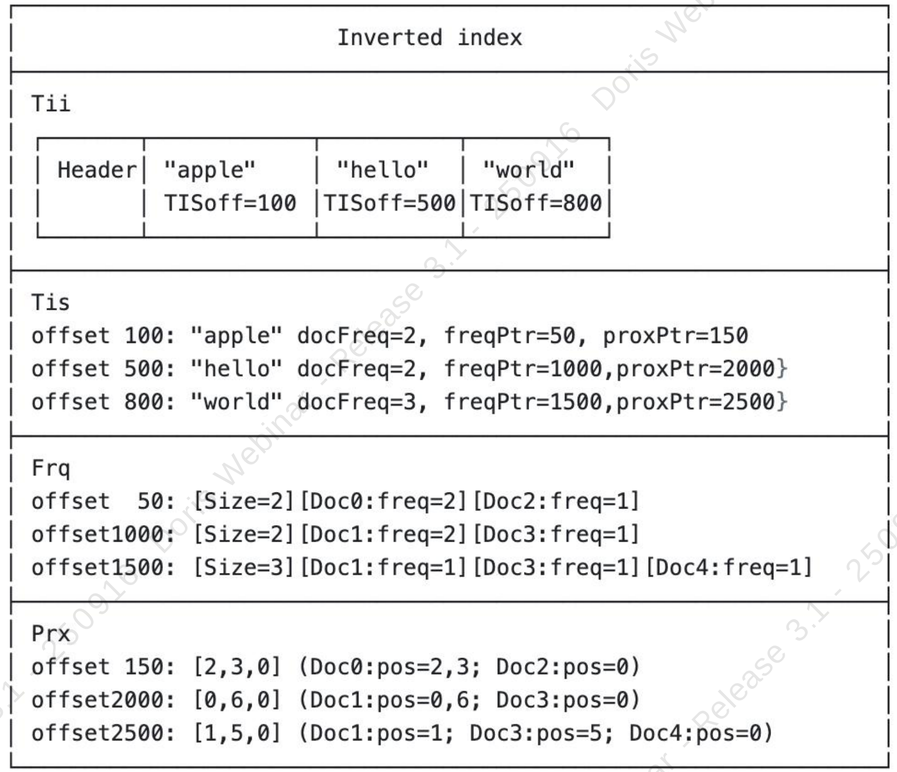
- 独立于数据文件dat
- 有利于独立管理索引，比如轻量级构建索引，单独进行索引合并等
- 增加文件管理成本

存储格式

- V1，每个索引一个文件
- V2，所有索引合并成一个文件
- V3，压缩词典文件和位置信息文件

Doris倒排索引存储格式V3压缩原理

文件布局示例：



举例

- 假设我们有以下文档集合和三个terms: "apple", "hello", "world"
- Doc0: "I like apple juice and apple pie
- Doc1: "Hello world, this is a hello message
- Doc2: "Apple trees grow in the garden"
- Doc3: "Hello everyone, welcome to the world"
- Doc4: "World peace is important for everyone"

- apple: {Doc0(freq=2), Doc2(freq=1)} → docFreq=2
- hello: {Doc1(freq=2), Doc3(freq=1)} → docFreq=2
- world: {Doc1(freq=1), Doc3(freq=1), Doc4(freq=1)} → docFreq=3

压缩优化

- 对Prx文件中每个term对应的位置信息list采用PFOR压缩
- 对Tis词典采用ZSTD压缩

PFOR (Patched Frame of Reference)

- 专门针对倒排索引中整数序列设计的压缩算法。
- 基本思想：大部分整数用较少的位数存储，少数异常值单独处理
- 原始文档ID增量序列：[1, 2, 1, 3, 1, 2, 1, 500, 1, 2]
- 压缩：大部分值 ≤ 3 (2位可存储)，异常值：500

Doris倒排索引存储格式V3使用方式

```
CREATE TABLE example_table (
    content TEXT,
    INDEX content_idx (content) USING
INVERTED
    PROPERTIES("parser" = "english",
"dict_compression" = "true")
) ENGINE=OLAP
PROPERTIES
("inverted_index_storage_format" = "V3");
```

- 建表语句的properties字段配置inverted_index_storage_format“ = ”V3“
- 同时倒排索引properties可以指定“dict_compression” = “true”开启字典压缩
- 如果需要对数据文件本身进一步提升压缩效果的话，可以在properties字段中修改storage_page_size大小，默认为64k，比如可以调整为512k

Doris倒排索引存储格式V3压缩效果

- 在标准测试集httplogs和logsbench上，开启V3格式能达到20%以上的索引文件压缩空间节省效果。
- 在某客户真实场景下，V3更是达到50%以上空间节省。
- 适合大规模文本数据、日志分析场景，尤其是对磁盘存储成本较为在意的场景。

测试集	导入前数据大小	inverted_index_storage_format = v2	inverted_index_storage_format = v3	v3 较 v2 空间节省
httplogs	30.89 GB	4.472 GB	3.479 GB	22.2%
logsbench	1479.31 GB	182.180 GB	138.008 GB	24.2%

目录

倒排索引存储格式V3

新增三种内置分词器

自定义分词能力

新增三种内置分词器

ICU分词器

- ICU（International Components for Unicode）
- 适用场景：包含复杂文字系统的国际化文本，特别适合多语言混合文档。

```
SELECT TOKENIZE(' حبا ب½العالمHello 世界', '"parser"="icu"');
-- 结果: ["", ","العالم½ð" ,"حباHello", "世界"]

SELECT TOKENIZE('มนไมเปนไปตามความตองการ', '"parser"="icu"');
-- 结果: ["มน", "ไมเปน", "ไป", "ตาม", "ความ", "ตองการ"]
```

IK分词器

- 基于算法的高级中文分词，结合词典和统计模型
- 适用场景：对分词质量要求较高的中文文本处理
- ik_smart：智能模式，词少且更长，语义集中，适合精确搜索
- ik_max_word：最细粒度模式，更多短词，覆盖更全面，适合召回搜索

```
SELECT TOKENIZE('中华人民共和国国歌',
'"parser"="ik","parser_mode"="ik_smart"');
-- 结果: ["中华人民共和国", "国歌"]

SELECT TOKENIZE('中华人民共和国国歌',
'"parser"="ik","parser_mode"="ik_max_word"');
-- 结果: ["中华人民共和国", "中华人民", "中华", "华人", "人民共和国", "人民", "共和国", "共和", "国歌"]
```

新增三种内置分词器

Basic分词器

- 实现：采用字符类型识别进行分词
- 连续的字母数字字符作为一个词（word tokens）
- 中文字符单独分词（每个汉字一个 token）
- 忽略标点符号、空格和特殊符号
- 适用场景：简单场景、对性能要求极高的场景，**可以作为日志场景中unicode分词器的平替**

```
-- 英文文本分词
SELECT TOKENIZE('Hello World! This is a test.', '"parser"="basic"');
-- 结果: ["hello", "world", "this", "is", "a", "test"]

-- 中文文本分词
SELECT TOKENIZE('你好世界', '"parser"="basic"');
-- 结果: ["你", "好", "世", "界"]

-- 混合语言分词
SELECT TOKENIZE('Hello你好World世界', '"parser"="basic"');
-- 结果: ["hello", "你", "好", "world", "世", "界"]

-- 包含数字和特殊字符
SELECT TOKENIZE('GET /images/hm_bg.jpg HTTP/1.0',
'"parser"="basic"');
-- 结果: ["get", "images", "hm", "bg", "jpg", "http", "1", "0"]

-- 处理长数字序列
SELECT TOKENIZE('12345678901234567890', '"parser"="basic"');
-- 结果: ["12345678901234567890"]
```

目录

为什么要引入自定义分词

提升用户文本检索的召回率

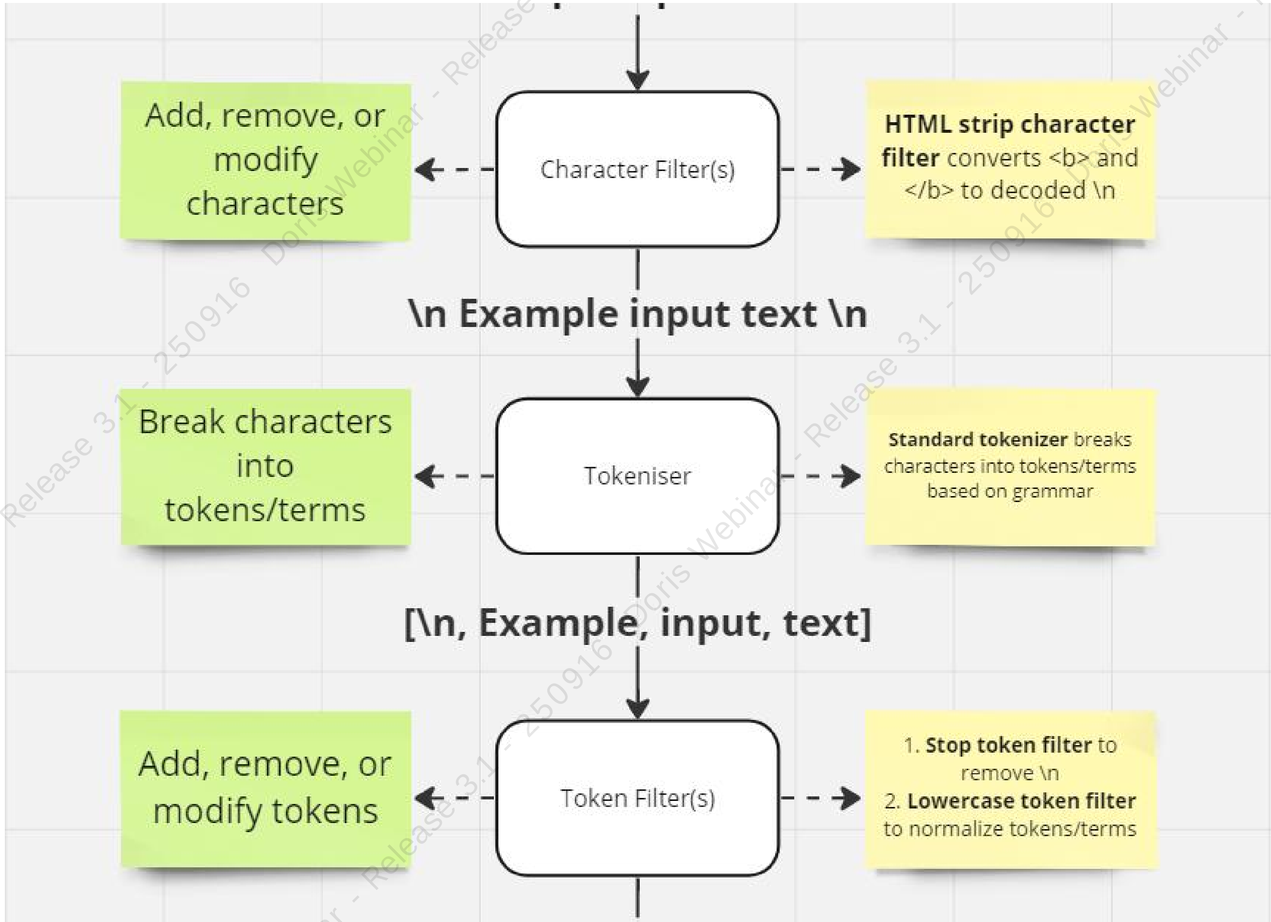
- 内置分词器无法完全满足用户的分词需求，继而影响文本检索的召回率，但是重新开发一种新的分词器代价过高。
- 用户有对分词器进行精细化定制需求，比如自定义分隔符。

个性化的文本检索需求

- 一些用户的实际业务场景往往要求对不同类型的字段采用差异化的文本处理策略，单一的内置 analyzer 很难覆盖

自定义分词的构成

- 管道化组合：通过 char filter、tokenizer 与多个 token filter 的链式配置，构建自定义文本处理流程。
- 组件复用：常用的 tokenizer 和 filter 可在多个 analyzer 中共享，减少重复定义，降低维护成本。
- 用户可以通过Doris 3.1版本提供的自定义分词机制，支持灵活组合char filter、tokenizer 和 token filter，从而为不同字段定制合适的分词流程，满足复杂场景下的个性化文本检索需求。



自定义分词使用举例

- 用户使用unicode/standard分词器的时候，遇到apy217.39_202501260000026_526这种文本，无法通过match(apy217)或者match(202501260000026)匹配中对应文本，只能通过match(apy217.39_202501260000026_526)完整才能匹配。

```
-- 1. 创建自定义 token filter
CREATE INVERTED INDEX TOKEN_FILTER IF NOT EXISTS
complex_word_splitter
PROPERTIES
(
    "type" = "word_delimiter",
    "type_table" = "[. => SUBWORD_DELIM], [_ =>
SUBWORD_DELIM]");

-- 2. 创建自定义分词器
CREATE INVERTED INDEX ANALYZER IF NOT EXISTS
complex_identifier_analyzer
PROPERTIES
(
    "tokenizer" = "standard",
    "token_filter" = "complex_word_splitter, lowercase"
);
```

- 创建类型为word_delimiter的token filter，通过配置 Word Delimiter Filter，将点号(.)和下划线(_)设置为分隔符。
- 创建自定义分词器complex_identifier_analyzer，引用token filter complex_word_splitter。
- SELECT TOKENIZE('apy217.39_202501260000026_526', '"analyzer"="complex_identifier_analyzer"');结果为[apy]、[217]、[39]、[202501260000026]、[526]
- MATCH('apy217')或者MATCH('202501260000026')都可以命中

自定义分词使用举例

- 用户想把多值列通过特殊字符拼接为一个单值列，然后通过倒排索引的match去匹配其中的任意列字符，类似：拼接姓名|手机号|公司，alice|123456|company。

```
-- 创建用于多值列分割的 char group tokenizer
CREATE INVERTED INDEX TOKENIZER IF NOT EXISTS
multi_value_tokenizer
PROPERTIES
(
    "type" = "char_group",
    "tokenize_on_chars" = "[]",
    "max_token_length" = "255"
);
-- 创建多值列分词器
CREATE INVERTED INDEX ANALYZER IF NOT EXISTS
multi_value_analyzer
PROPERTIES
(
    "tokenizer" = "multi_value_tokenizer",
    "token_filter" = "lowercase, asciifolding"
);
```

- 创建类型为char_group的tokenizer multi_value_tokenizer，只将符号|设置为分隔符。
- 创建自定义分词器multi_value_analyzer，引用tokenizer multi_value_tokenizer 。
- SELECT tokenize('alice|123456|company', '"analyzer"="multi_value_analyzer"');
- 结果为[alice]、[123446]、[company]
- MATCH_ANY(‘alice’)或者MATCH_ANY(‘123456’)都可以命中

Q&A