知乎 首发于 Elasticsearch技术研讨

Lucene解析 - 基本概念

木洛
招贤纳士，欢迎自荐！

取消关注

83 人赞同了该文章

前言

Apache Lucene是一个开源的高性能、可扩展的信息检索引擎，提供了强大的数据检索能力。Lucene已经发展了很多年，其功能越来越强大，架构也越来越精细。它目前不仅仅能支持全文索引，也能够提供多种其他类型的索引方式，来满足不同类型的查询需求。

基于Lucene的开源项目有很多，最知名的要属Elasticsearch和Solr，如果说Elasticsearch和Solr是一辆设计精美、性能卓越的跑车，那Lucene就是为其提供强大动力的引擎。为了驾驭这辆跑车让它跑的更快更稳定，我们需要对它的引擎研究透彻。

在此之前我们在专栏已经发表了多篇文章来剖析Elasticsearch的数据模型、读写路径、分布式架构以及Data/Meta一致性等问题，这篇文章之后我们会陆续发表一系列的关于Lucene的原理和源码解读，来全面解析Lucene的数据模型和数据读写路径。

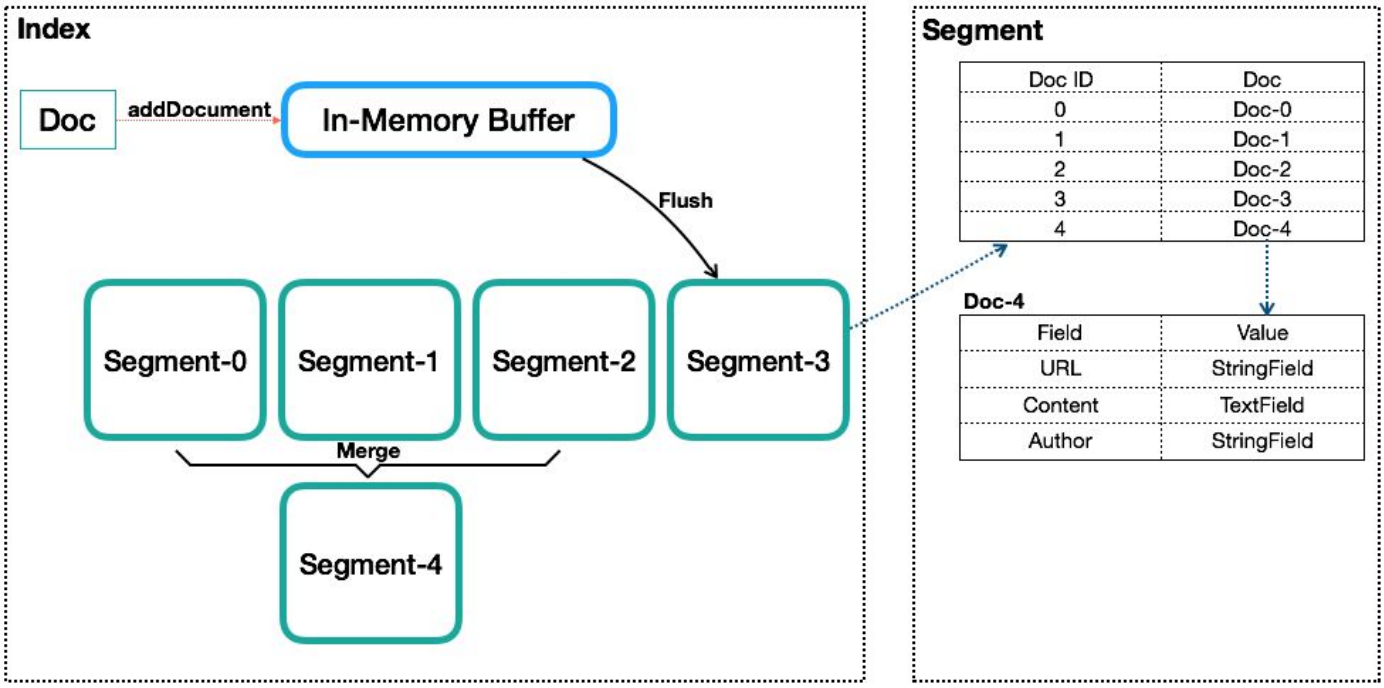
Lucene官方对自己的优势总结为几点：

1. Scalable, High-Performance Indexing
2. Powerful, Accurate and Efficient Search Algorithms

整个分析会基于Lucene 7.2.1版本，在读这篇文章之前，需要有一定的知识基础，例如了解基本的搜索和索引原理，知道什么是倒排、分词、相关性等基本概念，了解Lucene的基本使用，例如Directory、IndexWriter、IndexSearcher等。

基本概念

在深入解读Lucene之前，先了解下Lucene的几个基本概念，以及这几个概念背后隐藏的一些东西。



抽象架构图

Index（索引）

类似数据库的表的概念，但是与传统表的概念会有很大的不同。传统关系型数据库或者NoSQL数据库的表，在创建时至少要定义表的Scheme，定义表的主键或列等，会有一些明确定义的约束。而Lucene的Index，则完全没有约束。Lucene的Index可以理解为一个文档收纳箱，你可以往内部塞入新的文档，或者从里面拿出文档，但如果你要修改里面的某个文档，则必须先拿出来修改后再塞回去。这个收纳箱可以塞入各种类型的文档，文档里的内容可以任意定义，Lucene都能对其进行索引。

Document（文档）

类似数据库内的行或者文档数据库内的文档的概念，一个Index内会包含多个Document。写入Index的Document会被分配一个唯一的ID，即Sequence Number（更多被叫做DocId），

Field（字段）

一个Document会由一个或多个Field组成，Field是Lucene中数据索引的最小定义单位。Lucene提供多种不同类型的Field，例如StringField、TextField、LongFiled或NumericDocValuesField等，Lucene根据Field的类型（FieldType）来判断该数据要采用哪种类型的索引方式（Invert Index、Store Field、DocValues或N-dimensional等），关于Field和FieldType后面会再细说。

Term和Term Dictionary

Lucene中索引和搜索的最小单位，一个Field会由一个或多个Term组成，Term是由Field经过Analyzer（分词）产生。Term Dictionary即Term词典，是根据条件查找Term的基本索引。

Segment

一个Index会由一个或多个sub-index构成，sub-index被称为Segment。Lucene的Segment设计思想，与LSM类似但又有些不同，继承了LSM中数据写入的优点，但是在查询上只能提供近实时而非实时查询。

Lucene中的数据写入会先写内存的一个Buffer（类似LSM的MemTable，但是不可读），当Buffer内数据到一定量后会被flush成一个Segment，每个Segment有自己独立的索引，可独立被查询，但数据永远不能被更改。这种模式避免了随机写，数据写入都是Batch和Append，能达到很高的吞吐量。Segment中写入的文档不可被修改，但可被删除，删除的方式也不是在文件内部原地更改，而是会由另外一个文件保存需要被删除的文档的DocID，保证数据文件不可被修改。Index的查询需要对多个Segment进行查询并对结果进行合并，还需要处理被删除的文档，为了对查询进行优化，Lucene会有策略对多个Segment进行合并，这点与LSM对SSTable的Merge类似。

Segment在被flush或commit之前，数据保存在内存中，是不可被搜索的，这也就是为什么Lucene被称为提供近实时而非实时查询的原因。读了它的代码后，发现它并不是不能实现数据写入即可查，只是实现起来比较复杂。原因是Lucene中数据搜索依赖构建的索引（例如倒排依赖Term Dictionary），Lucene中对数据索引的构建会在Segment flush时，而非实时构建，目的是为了构建最高效索引。当然它可引入另外一套索引机制，在数据实时写入时即构建，但这套索引实现会与当前Segment内索引不同，需要引入额外的写入时索引以及另外一套查询机制，有一定复杂度。

Sequence Number

Sequence Number（后面统一叫DocId）是Lucene中一个很重要的概念，数据库内通过主键来唯一标识一行，而Lucene的Index通过DocId来唯一标识一个Doc。不过有几点要特别注意：

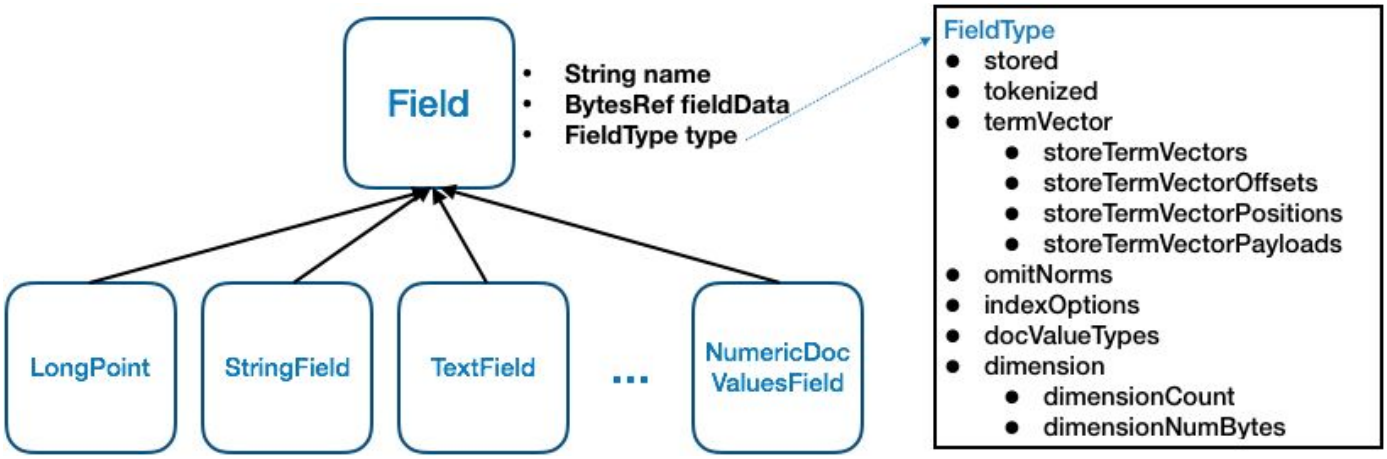
案很简单，Segment之间是有顺序的，举个简单的例子，一个Index内有两个Segment，每个Segment内分别有100个Doc，在Segment内DocId都是0-100，转换到Index级的DocId，需要将第二个Segment的DocId范围转换为100-200。
2. DocId在Segment内唯一，取值从0开始递增。但不代表DocId取值一定是连续的，如果有Doc被删除，那可能会存在空洞。
3. 一个文档对应的DocId可能会发生变化，主要是发生在Segment合并时。

Lucene内最核心的倒排索引，本质上就是Term到所有包含该Term的文档的DocId列表的映射。所以Lucene内部在搜索的时候会是一个两阶段的查询，第一阶段是通过给定的Term的条件找到所有Doc的DocId列表，第二阶段是根据DocId查找Doc。Lucene提供基于Term的搜索功能，也提供基于DocId的查询功能。

DocId采用一个从0开始底层的Int32值，是一个比较大的优化，同时体现在数据压缩和查询效率上。例如数据压缩上的Delta策略、ZigZag编码，以及倒排列表上采用的SkipList等，这些优化后续会详述。

索引类型

Lucene中支持丰富的字段类型，每种字段类型确定了支持的数据类型以及索引方式，目前支持的字段类型包括LongPoint、TextField、StringField、NumericDocValuesField等。



Lucene Field

如图是Lucene中对于不同类型Field定义的一个基本关系，所有字段类都会继承自Field这个类，Field包含3个重要属性：name(String)、fieldsData(BytesRef)和type(FieldType)。name即字段的名称，fieldsData即字段值，所有类型的字段的值最终都会转换为二进制字节流来表示。type是字段类型，确定了该字段被索引的方式。

FieldType是一个很重要的类，包含多个重要属性，这些属性的值决定了该字段被索引的方式

数据以及组合FieldType内索引参数来达到定制类型的目的。

要理解Lucene能够提供哪些索引方式，只需要理解FieldType内每个属性的具体含义，我们来一个一个看：

- **stored**: 代表是否需要保存该字段，如果为false，则lucene不会保存这个字段的值，而搜索结果中返回的文档只会包含保存了的字段。
- **tokenized**: 代表是否做分词，在lucene中只有TextField这一个字段需要做分词。
- **termVector**: 这篇文章很好的解释了term vector的概念，简单来说，term vector保存了一个文档内所有的term的相关信息，包括Term值、出现次数（frequencies）以及位置（positions）等，是一个per-document inverted index，提供了根据docid来查找该文档内所有term信息的能力。对于长度较小的字段不建议开启term verctor，因为只需要重新做一遍分词即可拿到term信息，而针对长度较长或者分词代价较大的字段，则建议开启term vector。Term vector的用途主要有两个，一是关键词高亮，二是做文档间的相似度匹配（more-like-this）。
- **omitNorms**: Norms是normalization的缩写，lucene允许每个文档的每个字段都存储一个normalization factor，是和搜索时的相关性计算有关的一个系数。Norms的存储只占一个字节，但是每个文档的每个字段都会独立存储一份，且Norms数据会全部加载到内存。所以若开启了Norms，会消耗额外的存储空间和内存。但若关闭了Norms，则无法做index-time boosting（elasticsearch官方建议使用query-time boosting来替代）以及length normalization。
- **indexOptions**: Lucene提供倒排索引的5种可选参数（NONE、DOCS、DOCS_AND_FREQS、DOCS_AND_FREQS_AND_POSITIONS、DOCS_AND_FREQS_AND_POSITIONS_AND_OFFSETS），用于选择该字段是否需要被索引，以及索引哪些内容。
- **docValuesType**: DocValue是Lucene 4.0引入的一个正向索引（docid到field的一个列存），大大优化了sorting、faceting或aggregation的效率。DocValues是一个强schema的存储结构，开启DocValues的字段必须拥有严格一致的类型，目前Lucene只提供NUMERIC、BINARY、SORTED、SORTED_NUMERIC和SORTED_SET五种类型。
- **dimension**：Lucene支持多维数据的索引，采取特殊的索引来优化对多维数据的查询，这类数据最典型的应用场景是地理位置索引，一般经纬度数据会采取这个索引方式。

来看下Lucene中对StringField的一个定义：

```java
/** Indexed, not tokenized, omits norms, indexes
 *  DOCS_ONLY, not stored. */
public static final FieldType TYPE_NOT_STORED = new FieldType();

/** Indexed, not tokenized, omits norms, indexes
 *  DOCS_ONLY, stored */
public static final FieldType TYPE_STORED = new FieldType();

static {
  TYPE_NOT_STORED.setOmitNorms(true);
  TYPE_NOT_STORED.setIndexOptions(IndexOptions.DOCS);
  TYPE_NOT_STORED.setTokenized(false);
  TYPE_NOT_STORED.freeze();

  TYPE_STORED.setOmitNorms(true);
  TYPE_STORED.setIndexOptions(IndexOptions.DOCS);
  TYPE_STORED.setStored(true);
  TYPE_STORED.setTokenized(false);
  TYPE_STORED.freeze();
}
```

Lucene StringField

StringFiled有两种类型索引定义，TYPE_NOT_STORED和TYPE_STORED，唯一的区别是这个Field是否需要Store。从其他的几个属性也可以解读出，StringFiled选择omitNorms，需要进行倒排索引并且不需要被分词。

Elasticsearch数据类型

Elasticsearch内对用户输入文档内Field的索引，也是按照Lucene能提供的几种模式来提供。除了用户能自定义的Field，Elasticsearch还有自己预留的系统字段，用作一些特殊的目的。这些字段映射到Lucene本质上也是一个Field，与用户自定义的Field无任何区别，只不过Elasticsearch根据这些系统字段不同的使用目的，定制有不同的索引方式。

```
public static class Defaults {

    public static final String NAME = VersionFieldMapper.NAME;
    public static final MappedFieldType FIELD_TYPE = new VersionFieldType();

    static {
        FIELD_TYPE.setName(NAME);
        FIELD_TYPE.setDocValuesType(DocValuesType.NUMERIC);
        FIELD_TYPE.setIndexOptions(IndexOptions.NONE);
        FIELD_TYPE.setHasDocValues(true);
        FIELD_TYPE.freeze();
    }
}
```

```
public static class Defaults {
    public static final String NAME = UidFieldMapper.NAME;

    public static final MappedFieldType FIELD_TYPE = new UidFieldType();
    public static final MappedFieldType NESTED_FIELD_TYPE;

    static {
        FIELD_TYPE.setIndexOptions(IndexOptions.DOCS);
        FIELD_TYPE.setTokenized(false);
        FIELD_TYPE.setStored(true);
        FIELD_TYPE.setOmitNorms(true);
        FIELD_TYPE.setIndexAnalyzer(Lucene.KEYWORD_ANALYZER);
        FIELD_TYPE.setSearchAnalyzer(Lucene.KEYWORD_ANALYZER);
        FIELD_TYPE.setName(NAME);
        FIELD_TYPE.freeze();

        NESTED_FIELD_TYPE = FIELD_TYPE.clone();
        NESTED_FIELD_TYPE.setStored(false);
        NESTED_FIELD_TYPE.freeze();
    }
}
```

Elasticsearch Field

举个例子，上图是Elasticsearch内两个系统字段_version和_uid的FieldType定义，我们来解读下它们的索引方式。Elasticsearch通过_uid字段唯一标识一个文档，通过_version字段来记录该文档当前的版本。从这两个字段的FieldType定义上可以看到，_uid字段会做倒排索引，不需要分词，需要被Store。而_version字段则不需要被倒排索引，也不需要被Store，但是需要被正排索引。很好理解，因为_uid需要被搜索，而_version不需要。但_version需要通过docId来查询，而且Elasticsearch内versionMap内需要通过docId做大量查询且只需要查询出_version字段，所以_version最合适的是被正排索引。

关于Elasticsearch内系统字段全面的解析，可以看下这篇文章。

总结

这篇文章主要介绍了Lucene的一些基本概念以及提供的索引类型。后续我们会有一系列文章来解析Lucene提供的IndexWriter的写入流程，其In-Memory Buffer的结构以及持久化后的索引文件结构，来了解Lucene为何能达到如此高效的数据索引性能。也会去解析IndexSearcher的查询流程，以及一些特殊的查询优化的数据结构，来了解为何Lucene能提供如此高效的搜索和查询。

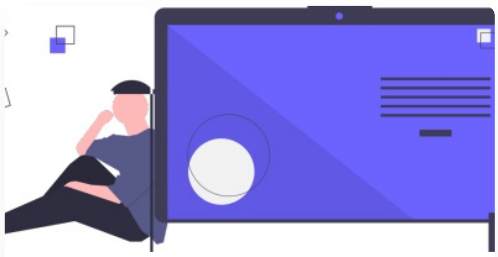
本文首发于云栖社区(Lucene解析 - 基本概念-博客-云栖社区-阿里云)，由原作者转载。

编辑于 2018-04-09

编程 NoSQL Lucene

文章被以下专栏收录

推荐阅读

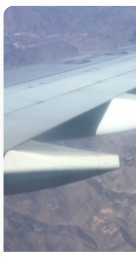


Elasticsearch从入门到放弃：文档CRUD要牢记

Jackey

从 Lucene 到 Elasticsearch

柳树 发表于Beaut...



Elastics 模型篇

少强

12 条评论

切换为时间排序

写下你的评论...

Golion 2018-04-09

好文。

赞

ScriptShi 2018-04-10

“一个Index内有两个Segment，每个Segment内分别有100个Doc，在Segment内DocId都是0-100，转换到Index级的DocId，需要将第二个Segment的DocId范围转换为100-200。”

转化如果是实时的话，耗费cpu时间，如果是提前算好存下来的话，存多个级别的id浪费空间。

麻烦问一下博主相对于segmentid和idnexid一样，这样去做转化的好处是什么呢？ 还是因为有什么特别的用处吗？

赞

Segment内doc的起始ID都是0，所以必须做这个转换。为啥起始一定要是0，因为这样实现比更清晰。

赞

ScriptShi 回复 木洛 (作者) 2018-04-10

了解了！文章写得很nice啊

赞

张亮 2018-04-21

你好，我是滴滴这块负责elasticsearch平台化建设的童鞋，看到你们近期一直在做这个方面研究，我们在杭州，是否方便做线下交流?

赞

木洛 (作者) 回复 张亮 2018-04-21

你好，私聊。

赞

bigquestion 2018-05-14

"当然它可引入另外一套索引机制，在数据实时写入时即构建，但这套索引实现会与当前Segment内索引不同，需要引入额外的写入时索引以及另外一套查询机制，有一定复杂度。" 这个 可否细说一下

赞

木洛 (作者) 回复 bigquestion 2018-05-15

简单说就是能够让内存中的segment可被检索，实现起来不容易。

赞

海原星宿 2018-05-26

引 "Lucene中的数据写入会先写内存的一个Buffer（类似LSM的MemTable，但是不可读），当Buffer内数据到一定量后会被flush成一个Segment，每个Segment有自己独立的索引，可独立被查询，但数据永远不能被更改。" -----Segment中的数据为什么不能被更改？是因为更改的话会造成重新索引吗?

赞

不得了啦 回复 海原星宿 2019-07-10

个人理解更改已有文件会造成文件随机读写，影响系统的吞吐量

海原星宿 回复 不得了啦 2019-10-19

segement被flush到磁盘之后才是searchable的？那如果Buffer内的数据等了好久也没达到阈值，压根没机会被flush该怎么办？从而也就不可读了

赞

王泽宇 2019-12-17

我记得es refresh后生成segment就可读了，半个小时才flush一次，文中说：Lucene中的数据写入会先写内存的一个Buffer（类似LSM的MemTable，但是不可读），当Buffer内数据到一定量后会被flush成一个Segment，这两个是否冲突，有点晕

赞