

知乎

首发于
Elasticsearch技术研讨

Elasticsearch内核解析 - 数据模型篇



少强

招聘应届生，社招，求私信

关注他

37 人赞同了该文章

Elasticsearch是一个实时的分布式搜索和分析引擎，它可以帮助我们很快的速度去处理大规模数据，可以用于全文检索、结构化检索、推荐、分析以及统计聚合等多种场景。

Elasticsearch是一个建立在全文搜索引擎库Apache Lucene 基础上的分布式搜索引擎，Lucene最早的版本是2000年发布的，距今已经18年，是当今最先进，最高效的全功能开源搜索引擎框架，众多搜索领域的系统都基于Lucene开发，比如Nutch，Solr和Elasticsearch等。Elasticsearch第一个版本发布于2010年，发布后就以非常快的速度霸占了开源搜索系统领域，成为目前搜索领域的首选，著名的维基百科，GitHub和Stack Overflow都在使用它。

既然有Lucene，为啥还会出现很火的Elasticsearch？回答这个问题之前，我们先来简单看看Lucene中的一些数据模型：



Lucene中包含了四种基本数据类型，分别是：

- Index：索引，由很多的Document组成。
- Document：由很多的Field组成，是Index和Search的最小单位。
- Field：由很多的Term组成，包括Field Name和Field Value。
- Term：由很多的字节组成，可以分词。

上述四种类型在Elasticsearch中同样存在，意思也一样。

Lucene中存储的索引主要分为三种类型：

- Invert Index：倒排索引，或者简称Index，通过Term可以查询到拥有该Term的文档。可以配置为是否分词，如果分词可以配置不同的分词器。索引存储的时候有多种存储类型，分别是：
 - DOCS：只存储DocID。
 - DOCS_AND_FREQS：存储DocID和词频（Term Freq）。
 - DOCS_AND_FREQS_AND_POSITIONS：存储DocID、词频（Term Freq）和位置。
 - DOCS_AND_FREQS_AND_POSITIONS_AND_OFFSETS：存储DocID、词频（Term Freq）、位置和偏移。
- DocValues：正排索引，采用列式存储。通过DocID可以快速读取到该Doc的特定字段的值。由于是列式存储，性能会比较好。一般用于sort，agg等需要高频读取Doc字段值的场景。
- Store：字段原始内容存储，同一篇文章的多个Field的Store会存储在一起，适用于一次读取少量且多个字段内存的场景，比如摘要等。

Lucene中提供索引和搜索的最小组织形式是Segment，Segment中按照索引类型不同，分成了Invert Index，Doc Values和Store这三大类（还有一些辅助类，这里省略），每一类里面都是按照Doc为最小单位存储。Invert Index中存储的Key是Term，Value是Doc ID的链表；Doc Value中Key是Doc ID和Field Name，Value是Field Value；Store的Key是Doc ID，Value是Filed Name和Filed Value。

由于Lucene中没有主键概念和更新逻辑，所有对Lucene的更新都是Append一个新Doc，类似于一个只能Append的队列，所有Doc都被同等对待，同样的处理方式。其中的Doc由众多Field组成，没有特殊Field，每个Field也都被同等对待，同样的处理方式。

从上面介绍来看，Lucene只是提供了一个索引和查询的最基本的功能，距离一个完全可用的完整搜索引擎还有一些距离：

Lucene的不足

1. Lucene是一个单机的搜索库，如何能以分布式形式支持海量数据？



4. 在稀疏列数据中，如何判断某些文档是否存在特定字段？
5. Lucene中生成完整Segment后，该Segment就不能再被更改，此时该Segment才能被搜索，这种情况下，如何做实时搜索？

上述几个问题，对于搜索而言都是至关重要的功能诉求，我们接下来看看Elasticsearch中是如何来解这些问题的。

Elasticsearch怎么做

在Elasticsearch中，为了支持分布式，增加了一个系统字段_routing（路由），通过_routing将Doc分发到不同的Shard，不同的Shard可以位于不同的机器上，这样就能实现简单的分布式了。

采用类似的方式，Elasticsearch增加了_id、_version、_source和_seq_no等等多个系统字段，通过这些Elasticsearch中特有的系统字段可以有效解决上述的几个问题，新增的系统字段主要是下列几个：

ES System Field	含义	Lucene Index	Lucene DocValues	Lucene Store
_uid	主键	√		√
_version	版本		√	
_source	原文			√
_seq_no	顺序号	√	√	
_primary_term	Primary编号		√	
_routing	路由	√		√
_field_names	字段名	√		

下面我们逐个字段的剖析下上述系统字段的作用，先来看第一个_id字段：

1. _id



Lucene中没有主键索引，要保证系统中同一个Doc不会重复，Elasticsearch引入了_id字段来实现主键。每次写入的时候都会先查询id，如果有，则说明已经有相同Doc存在了。

通过_id值（ES内部转换成_uid）可以唯一在Elasticsearch中确定一个Doc。

Elasticsearch中，_id只是一个用户级别的虚拟字段，在Elasticsearch中并不会映射到Lucene中，所以也就不会存储该字段的值。

_id的值可以由_uid解析而来（_uid =type + '#' + id），Elasticsearch中会存储_uid。

2. _uid

_uid的格式是：type + '#' + id。

_uid会存储在Lucene中，在Lucene中的映射关系如下：dex下可能存在多个id值相同的Doc，而6.0.0之后只支持单Type，同Index下id值是唯一的。

uid会存储在Lucene中，在Lucene中的映射关系如下：

Field	Index	Index Type	Analyzer	DocValues	Store
_uid	Yes	Doc	No	No	Yes

_uid 只是存储了倒排Index和原文store：倒排Index的目的是可以通过_id快速查询到文档；原文store用来在返回的Response里面填充完整的_id值。

在Lucene中存储_uid，而不是_id的原因是，在6.0.0之前版本里面，_uid可以比_id表示更多的信息，比如Type。在6.0.0版本之后，同一个Index只能有一个Type，这时候Type就没多大意义了，后面Type应该会消失，那时候_id就会和_uid概念一样，到时候两者会合二为一，也能简化大家的理解。

3. _version

Elasticsearch中每个Doc都会有一个Version，该Version可以由用户指定，也可以由系统自动生成。如果是系统自动生成，那么每次Version都是递增1。



Version在Lucene中也是映射为一个特殊的Field存在。

Field	Index	Index Type	Analyzer	DocValues	Store
_Version	No	No	No	Yes	No

Elasticsearch中Version字段的主要目的是通过doc_id读取Version，所以Version只要存储为DocValues就可以了，类似于KeyValue存储。

Elasticsearch通过使用version来保证对文档的变更能以正确的顺序执行，避免乱序造成的数据丢失：

1. 首次写入Doc的时候，会为Doc分配一个初始的Version：V0，该值根据VersionType不同而不同。
2. 再次写入Doc的时候，如果Request中没有指定Version，则会先加锁，然后去读取该Doc的最大版本V1，然后将V1+1后的新版本写入Lucene中。
3. 再次写入Doc的时候，如果Request中指定了Version：V1，则继续会先加锁，然后去读该Doc的最大版本V2，判断V1==V2，如果不相等，则发生版本冲突。否则版本吻合，继续写入Lucene。
4. 当做部分更新的时候，会先通过GetRequest读取当前id的完整Doc和V1，接着和当前Request中的Doc合并为一个完整Doc。然后执行一些逻辑后，加锁，再次读取该Doc的最大版本号V2，判断V1==V2，如果不相等，则在刚才执行其他逻辑时被其他线程更改了当前文档，需要报错后重试。如果相等，则期间没有其他线程修改当前文档，继续写入Lucene中。这个过程就是一个典型的read-then-update事务。

4. _source

Elasticsearch中有一个重要的概念是source，存储原始文档，也可以通过过滤设置只存储特定Field。

Source在Lucene中也是映射为了一个特殊的Field存在：

Field	Index	Index Type	Analyzer	DocValues	Store
_source	No	No	No	No	Yes



_source其实是名为_source的虚拟Store Field。

Elasticsearch中使用_source字段可以实现以下功能：

- Update：部分更新时，需要从读取文档保存在_source字段中的原文，然后和请求中的部分字段合并为一个完整文档。如果没有_source，则不能完成部分字段的Update操作。
- Rebuild：最新的版本中新增了rebuild接口，可以通过Rebuild API完成索引重建，过程中不需要从其他系统导入全量数据，而是从当前文档的_source中读取。如果没有_source，则不能使用Rebuild API。
- Script：不管是Index还是Search的Script，都可能用到存储在Store中的原始内容，如果禁用了_source，则这部分功能不再可用。
- Summary：摘要信息也是来源于_source字段。


5. _seq_no

严格递增的顺序号，每个文档一个，Shard级别严格递增，保证后写入的Doc的_seq_no大于先写入的Doc的_seq_no。

任何类型的写操作，包括index、create、update和Delete，都会生成一个_seq_no。

_seq_no在Primary Node中由SequenceNumbersService生成，但其实真正产生这个值的是LocalCheckpointTracker，每次递增1：

```
/**
 * The next available sequence number.
 */
private volatile long nextSeqNo;
/**
 * Issue the next sequence number.
 *
 * @return the next assigned sequence number
 */
synchronized long generateSeqNo() {
    return nextSeqNo++;
}
```

每个文档在使用Lucene的document操作接口之前，会获取到一个_seq_no，这个_seq_no会以系统保留Field的名义存储到Lucene中，文档写入Lucene成功后，会标记该seq_no为完成状态 

checkpoint分为local_checkpoint和global_checkpoint，主要是用于保证有序性，以及减少Shard恢复时数据拷贝的数据拷贝量，更详细的介绍可以看这篇文章：[Sequence IDs: Coming Soon to an Elasticsearch Cluster Near You](#)。

_seq_no在Lucene中的映射：

Elasticsearch中_seq_no的作用有两个，一是通过doc_id查询到该文档的seq_no，二是通过seq_no范围查找相关文档，所以也就需要存储为Index和DocValues（或者Store）。由于是在冲突检测时才需要读取文档的_seq_no，而且此时只需要读取_seq_no，不需要其他字段，这时候存储为列式存储的DocValues比Store在性能上更好一些。

_seq_no是严格递增的，写入Lucene的顺序也是递增的，所以DocValues存储类型可以设置为Sorted。

另外，_seq_no的索引应该仅需要支持存储DocId就可以了，不需要FREQS、POSITIONS和分词。如果多存储了这些，对功能也没影响，就是多占了一点资源而已。

6. _primary_term

_primary_term也和_seq_no一样是一个整数，每当Primary Shard发生重新分配时，比如重启，Primary选举等，_primary_term会递增1。

_primary_term主要是用来恢复数据时处理当多个文档的_seq_no一样时的冲突，避免Primary Shard上的写入被覆盖。

Elasticsearch中_primary_term只需要通过doc_id读取到即可，所以只需要保存为DocValues就可以了。

7. _routing



在mapping中，或者Request中可以指定按某个字段路由。默认是按照_id值路由。

_routing在Lucene中映射为：

Elasticsearch中文档级别的_routing主要有两个目的，一是可以查询到使用某种_routing的文档有哪些，当发生_routing变化时，可以对历史_routing的文档重新读取再Index，这个需要倒排Index。另一个是查询到文档后，在Response里面展示该文档使用的_routing规则，这里需要存储为Store。

8. _field_names

该字段会索引某个Field的名称，用来判断某个Doc中是否存在某个Field，用于exists或者missing请求。

_field_names在Lucene中的映射：

Elasticsearch中_field_names的目的是查询哪些Doc的这个Field是否存在，所以只需要倒排Index即可。

总结

在上面的介绍中，我们解释了Elasticsearch是如何通过增加系统字段来扩充Lucene的功能，开篇提出的Lucene的多个不足中，前四个都在文章中做了说明，最后一个没法通过增加系统字段实现，我们将会在下篇《Elasticsearch写流程简介》中介绍如何通过其他方式来实现，下一篇见。

另外，我们招人：Elasticsearch和Lucene的开发，有兴趣的可以私信联系我。

编辑于 2018-04-11



赞赏

还没有人赞赏，快来当第一个赞赏的人吧！

Elasticsearch 搜索引擎 Lucene

文章被以下专栏收录

 **Elasticsearch技术研讨**
招 Elasticsearch 内核研发：base 杭州。

关注专栏

推荐阅读



elasticsearch
看完这篇还不会 Elasticsearch 搜索,那我就哭了!
武培轩 发表于后端技术社...


ES(Elasticsearch)支持PB级全文搜索引擎入门教程
全文搜索属于最常见的需求，开源的 Elasticsearch（以下简称 Elastic）是目前全文搜索引擎的首选。它可以快速地储存、搜索和分析海量数据。比如维基百科、Stack Overflow、Github 都采用它...
蛙课网

Elasticsearch
写在前面
Elasticsearch 一直停留有做深入简单的做 ES 的的一只菜鸟


12 条评论 切换为时间排序

写下你的评论...



 铭毅天下 2018-03-25

胡总，您好！看了您的系列Elasticsearch深入原理的文章，加强了我对ES底层的认知。想请教胡总，这块底层知识是怎么习得的？我自己也在研究相关Elasticsearch技术，已经有两年



知乎

首发于
Elasticsearch技术研讨

正所谓“授人以渔”，想请您介绍下相关底层原理的学习方法。

期待您的回复，万分感谢！

👍 2



少强 (作者) 回复 铭毅天下

2018-03-26

您好，我是毕业后就一直从事分布式系统的设计和研发，也参与过多个非常大型的分布式系统架构，积累了比较多的理论经验教训，而在分布式系统里面这些架构基本都是想通的，基本稍微看看就知道他们底层是怎么做的，有啥优缺点都是一目了然的，要学习的话，最好的方式就是加入我们团队，我们在大量招人做分布式系统，一起做分布式系统，基本就是天天讨论，设计，实现，验证，这样耳濡目染加参与会有非常大的帮助。如果没机会参与进来，那么就只有两种办法，一是在网上找分布式系统的资料文章自学，二是找一个开源分布式系统，然后看代码，如果看代码有困难，那么就找代码分析文章，跟着文章边看边读代码，或者是带着疑问去看，如果有多个人一起看更好，这样可以互相讨论，互相提问等等，但不管怎么，这都是一个长期的过程，不可能短期有收效，沉下心一步一步来，总有一天会融会贯通的，啰嗦了这么多，希望对您有用。

👍 13



kevin 回复 少强 (作者)

2018-04-01

具体的招聘要求呢，我研究过zk源码，elasticsearch应用层面

👍 赞

展开其他 1 条回复



我滴妈呀

2018-03-29

你好，6.0.0之后只支持单Type，我在运用的时候也发现了，我想知道为什么？看您的文章里没有解释？

👍 赞



少强 (作者) 回复 我滴妈呀

2018-03-29

一般情况下，两级结构的index+type其实没啥用，一般用一级index就够了，再多一个Type属于多此一举，后面版本中的type会直接去掉。

👍 1



我滴妈呀

2018-03-29

还有个问题，我是新手哈，看您对_version的介绍，是不是类似于一种乐观锁的思想呢？

👍 赞



少强 (作者) 回复 我滴妈呀

2018-03-29





chief

2018-04-25

您好，请问您这面的招聘要求是什么，之前因为工作或者兴趣研读过ES2.3.1, jstorm, hdfs部分源代码，对分布式系统比较感兴趣。

赞



少强 (作者) 回复 chief

2018-04-26

专栏里面有个置顶文章，里面有。

赞



键盘战神

2018-12-20

大神好，读了这篇文章之后收获不仅是文章内部的内容，感觉读源码的时候会更清晰。我遇到一个问题就是elasticsearch集群的打分是不停变化的，索引几乎没有变动，但是词频却在实时变化，请问这种是正常的嘛。

赞



river

2019-09-09

有了`_seqno`, `_version`是不是就没必要了

赞

