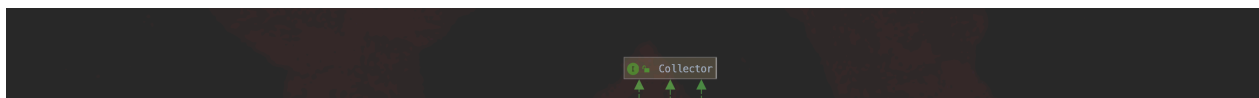


Collector (三)

本文承接[Collector \(二\)](#)，继续介绍其他的收集器。

图1是Lucene常用的几个Collector：

图1：



TopDocsCollector

TopFieldCollector

在[Collector \(二\)](#)的文章中，我们介绍了TopScoreDocCollector收集器以及它的两个子类SimpleTopScoreDocCollector、PagingTopScoreDocCollector，它们的排序规则是"先打分，后文档号"，TopFieldCollector的排序规则是"先域比较（[FieldComparator](#)），后文档号"。

- 先域比较（FieldComparator）：根据文档（Document）中的排序域（SortField）的域值进行排序。
- 后文档号：由于文档号是唯一的，所以当无法通过域比较获得顺序关系时，可以通过文档的文档号进行排序，文档号越小，排名越靠前（competitive）

我们先通过例子来介绍如何使用TopFieldCollector排序的例子，随后介绍排序的原理。

本人业务中常用的排序域有SortedNumericSortField、SortedSetSortField，其他的排序域可以看[SortField类](#)以及子类，在搜索阶段如果使用了域排序，那么Lucene默认使用TopFieldCollector来实现排序。

例子

SortedNumericSortField

图2：

```
// 文档0
Document doc = new Document();
doc.add(new TextField( name: "author", value: "aab b aab aabbcc ", Field.Store.YES));
```

图3:

```
SortField searchSortField = new SortedNumericSortField( field: "sortByNumber", SortField.Type.LONG);
```

- SortedNumericSortField: 根据文档 (document) 中NumericDocValuesField域的域值进行排序, 如果文档中没有这个域, 那么域值视为0
 - 图2为索引阶段的内容, 我们根据域名为“sortByNumber”的NumericDocValuesField域的域值进行排序, 其中文档0、文档4没有该域, 故它的域值被默认为0, 它们按照文档号排序
 - 图3为搜索阶段的内容, 使用SortedNumericSortField对结果进行排序, 所以按照从小到大排序 (图3中参数reverse为true的话, 那么结果按照从大到小排序), 那么排序结果为:

文档1 --> 文档0 --> 文档4 --> 文档3 --> 文档2

SortedSetSortField

图4:

```
// 文档0
Document doc = new Document();
doc.add(new TextField( name: "author", value: "aab b aab aabbcc ", Field.Store.YES));
doc.add(new TextField( name: "content", value: "a b", Field.Store.YES));
```

图5:

- SortedSetSortField: 根据文档 (document) 中NumericDocValuesField域的域值进行排序, 如果文档中没有这个域, 那么域值视为null, 被视为"最小"
 - 图3为索引阶段的内容, 允许设置相同域名的SortedSetSortField有多个域值, 这么做的好处在于, 在搜索阶段, 我们可以选择其中一个域值来进行排序, 提高了排序的灵活性
 - 图4为搜索阶段的内容, 使用域名为"sortByString"的SortedSetSortField域的域值进行排序, 其中文档0、文档4没有该域, 故它的域值被视为null, 它们之间按照文档号排序

如何在搜索阶段选择排序域值:

- 通过[SortedSetSelector.Tyte](#)来选择哪一个域值, SortedSetSelector提供了下面的参数
 - MIN: 选择域值最小的进行排序, 例如上图中文档1、文档2、文档3会分别使用域值"a"、"c"、"b"作为排序条件, 图5中即按照这个规则排序, 由于参数reverse为false, 所以排序结果从小到大排序, 其中文档0、文档4的排序域值为null:

文档0 --> 文档4 --> 文档1 --> 文档3 --> 文档2

- MAX: 选择域值最大的进行排序, 例如上图中文档1、文档2、文档3会分别使用域值"y"、"z"、"x"作为排序条件, 由于参数reverse为false, 所以排序结果从小到大排序, 其中文档0、文档4的排序域值为null:

文档0 --> 文档4 --> 文档3 --> 文档1 --> 文档2

- MIDDLE_MIN: 选择中间域值, 如果域值个数为偶数个, 那么中间的域值就有两个, 则取较小值, 例如上图中文档1、文档2、文档3会分别使用域值"f"、"e"、"d"作为排序条件, , 由于参数reverse为false, 所以排序结果从小到大排序, 其中文档0、文档4的排序域值为null:

文档0 --> 文档4 --> 文档3 --> 文档2 --> 文档1

- MIDDLE_MAX: 选择中间域值, 如果域值个数为偶数个, 那么中间的域值就有两个, 则取较大值, 例如上图中文档1、文档2、文档3会分别使用域值"h"、"i"、"j"作为排序条件, , 由于参数reverse为false, 所以排序结果从小到大排序, 其中文档0、文档4的排序域值为null:

文档0 --> 文档4 --> 文档1 --> 文档2 --> 文档3

SortedNumericSortField也可以在索引阶段设置多个具有相同域名的不同域值, 其用法跟**SortedSetSortField**一致, 不赘述。

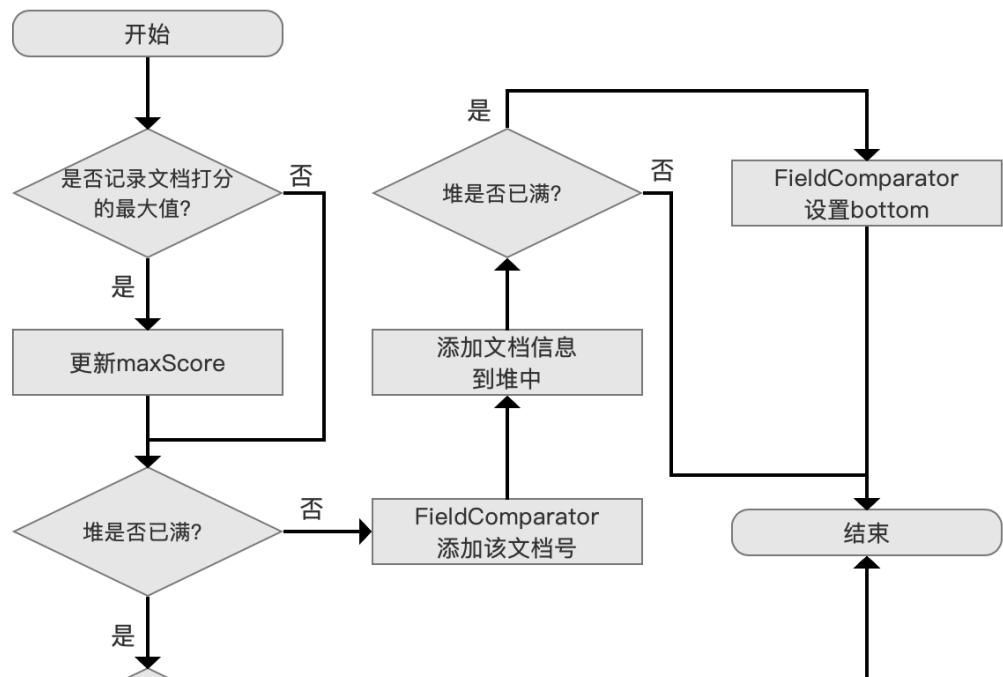
接下来我们根据过滤 (filtering) 规则, 我们接着介绍TopFieldCollector的两个子类:

- SimpleFieldCollector: 无过滤规则
- PagingFieldCollector: 有过滤规则, 具体内容在下文展开

SimpleFieldCollector

SimpleFieldCollector的collect(int doc)方法的流程图:

图6:



[点击查看大图](#)

预备知识

`IndexWriterConfig.IndexSort(Sort sort)`方法

在初始化IndexWriter对象时，我们需要提供一个IndexWriterConfig对象作为构造IndexWriter对象的参数，IndexWriterConfig提供了一个[setIndexSort\(Sort sort\)](#)的方法，该方法用来在索引期间按照参数Sort对象提供的排序规则对一个段内的文档进行排序，如果该排序规则跟搜索期间提供的排序规则（例如图3的排序规则）是一样的，那么很明显Collector收到的那些满足搜索条件的文档集合已经是有序的（因为Collector依次收到的文档号是从小到大有序的，而这个顺序描述了文档之间的顺序关系，下文会详细介绍）。

以下是一段进阶知识，需要看过[文档的增删改](#)以及[文档提交之flush](#)系列文章才能理解，看不懂可以跳过：

- 我们以图2作为例子，在单线程下（为了便于理解），如果不设置索引期间的排序或者该排序跟搜索期间的排序规则不一致，文档0~文档4对应的文档号分别是：0、1、2、3，Lucene会按照处理文档的顺序，分配一个从0开始递增的段内文档号，即[文档的增删改（下）（part 2）](#)中的numDocsInRAM，这是文档在一个段内的真实文档号，如果在索引期间设置了排序规则如下所示：

索引期间，图7

```
Analyzer analyzer = new WhitespaceAnalyzer();
IndexWriterConfig conf = new IndexWriterConfig(analyzer);
SortField indexSortField = new SortField( field: "sortByNumber", SortField.Type.LONG);
Sort indexSort = new Sort(indexSortField);
conf.setIndexSort(indexSort);
```

搜索期间，替换下图3的内容，使得图2中的例子中 搜索期间跟查询期间的有一样的排序规则，图8

```
DirectoryReader reader = DirectoryReader.open(indexWriter);
IndexSearcher indexSearcher = new IndexSearcher(reader);
SortField searchSortField = new SortField( field: "sortByNumber", SortField.Type.LONG);
Sort searchSort = new Sort(searchSortField);
indexSearcher.search(new MatchAllDocsQuery(), n: 2, searchSort);
```

- （重要）在图7、图8的代码条件下，传给Collector的文档号依旧分别是 0、1、2、3，但是这些文档号并不分别对应文档0~文档4了，根据排序规则，传给Collector的文档号docId跟文档编号的映射关系：

图9：

数组元素：文档编号	文档1	文档0	文档4	文档3	文档2
数组下标：docId	0	1	2	3	4

由图9可以知道，Collector还是依次收到 0 ~ 4的文档号，但是对应的文档号已经发生了变化，因为这些文档在索引期间已经根据域名为"sortByNumber"的SortedNumericSortField域的域值排好序了。

（极其重要）尽管在索引期间已经对段内的文档进行了排序，实际上文档0~文档4在段内的真实文档号依旧是：0、1、2、3，只是通过图9中的数组实现了映射关系，故给出下图：

图10:

图10中通过数组实现的映射关系即**Sorter.DocMap**对象**sortMap**，在flush阶段，生成**sortMap**（见[文档提交之flush（三）](#)）。

结语

TopFieldCollector相比较[Collector（二）](#)中TopScoreDocCollector，尽管他们都是TopDocsCollector的子类，由于存在**索引期间**的排序机制，使得TopFieldCollector的collect(int doc)的流程更加复杂，当然带来了更好的查询性能，至于如何能提高查询性能，由于篇幅原因，会在下一篇介绍图6的collect(int doc)的流程中展开介绍。

[点击](#)下载附件