

# FieldComparator && LeafFieldComparator

---

当满足搜索要求的文档被TopFieldCollector收集后，我们可以通过FieldComparator类来对这些结果（文档document）进行排序，并同时可以实现TopN的筛选。

## 排序类型

---

在介绍如果通过FieldComparator实现排序前，我们先介绍下排序类型，根据我们的排序对象，提供了下面几种类型。

### SCORE

---

根据文档打分(相关性 relevance)进行排序。打分越高越靠前。

### DOC

---

根据文档号进行排序，文档号越小越靠前

### STRING

---

根据String类型数据，即字符串对应的ord值(int类型)进行排序，ord越小越靠前(ord值的概念在[SortedDocValues](#)有详细介绍)。

### STRING\_VAL

---

根据String类型数据，即字符串进行排序，上文的STRING通过对应的ord进行排序，而在STRING\_VAL则是按照字典序逐个比较字符串的每一个字符。

通常情况下排序速度没有STRING快。比如只使用了[BinaryDocValues](#)的情况下，就只能使用这种排序类型，因为BinaryDocValues没有为String类型的域值设置ord值。

### INT

---

根据int类型数值进行排序，数值越小越靠前。

### FLOAT

---

根据float类型数值进行排序，数值越小越靠前。

### LONG

---

根据long类型数值进行排序，数值越小越靠前。

## DOUBLE

---

根据double类型数值进行排序，数值越小越靠前。

## CUSTOM

---

自定义类型，自定义实现一个实现FieldComparator类的子类。

## REWRITEABLE

---

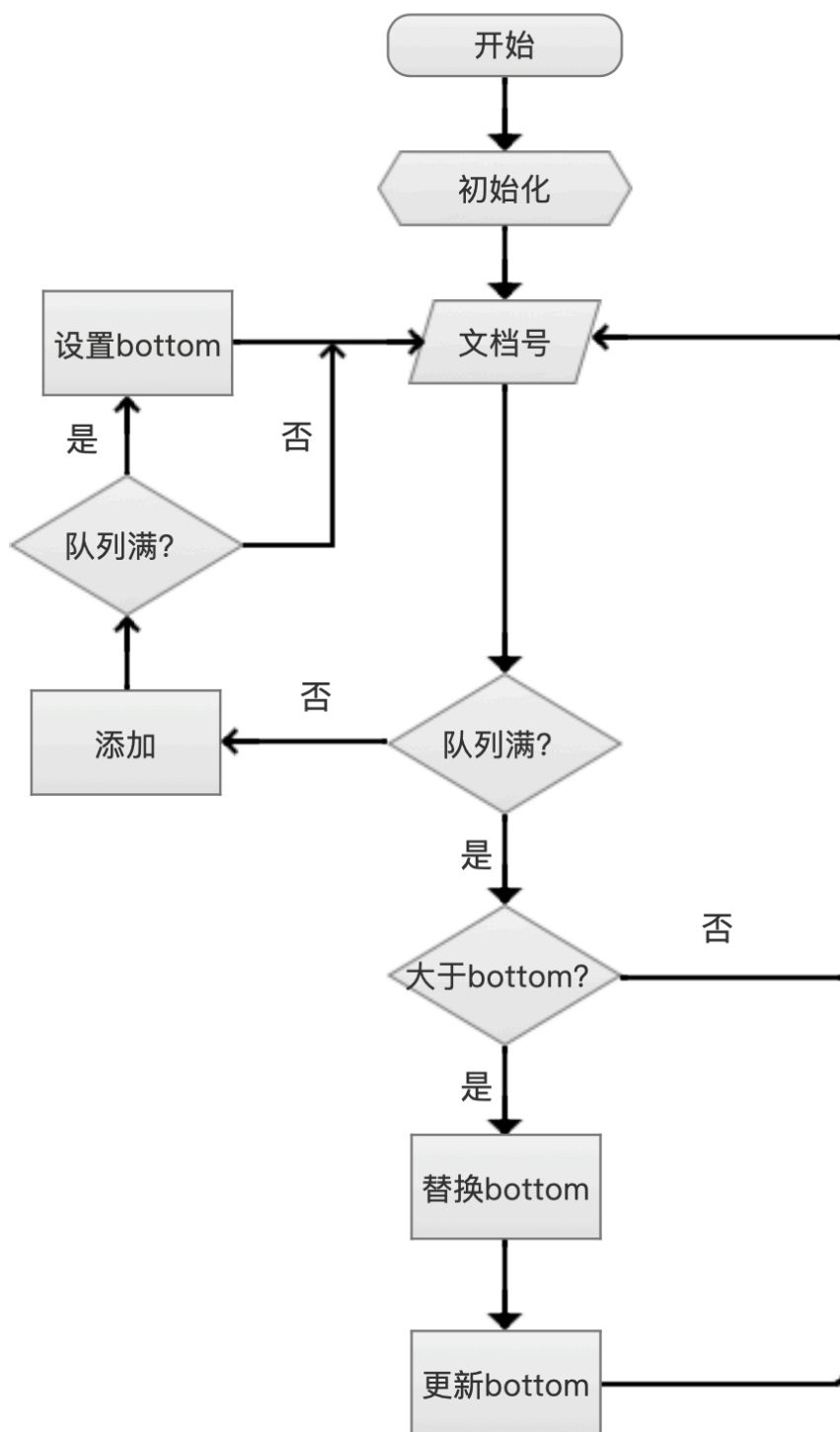
设置一个排序类型为可变的，使得每次搜索时候我们可以随时更新为新的排序规则。

## 流程图

---

我们一个例子来说明排序的过程，在这个例子中，我们使用STRING\_VAL来排序，即在索引阶段必须在每篇文档中定义BinaryDocValuesField，在介绍例子前，先介绍下排序的流程图。

图1：



初始化

图2:



在初始化阶段，根据设置TopN(必须设置)来初始化一个value[ ]数组，数组大小为N。value[ ]数组用来存放文档号对应的域值，在下面的例子中，即BinaryDocValuesField的域值，N的值为3。

## 文档号

图3：



由于比较的过程是在Collector类中的collect(int doc)中进行的，所以输入只能是满足搜索要求的文档号。

## 队列满？

图4：



即value[ ]数组中的元素个数是否等于数组的最大容量。

## 添加


图5：



根据文档号取出对应的BinaryDocValuesField的域值，将域值存放到value中。如何根据文档号取出域值在这里不赘述，如果你已经看过了[BinaryDocValues](#)这篇文章，那么就知过程啦。

## 设置bottom

图6：




```
graph TD; A[设置bottom] --> B{大于bottom?}; B --> C[替换bottom]; C --> D[更新bottom];
```

设置bottom

在添加的过程中，当队列满了，那么我们需要设置一个bottom值，bottom的值为数组中最小的那个（当前例子中按照字典序），设置bottom的目的在于，当有新的添加时，只要跟bottom做比较，如果大于bottom，那么就替换bottom，否则就直接跳过。

## 大于bottom?

图7:



```
graph TD; A[大于bottom?] --> B[替换bottom]; B --> C[更新bottom];
```

大于bottom?

判断新的添加是否大于bottom。

## 替换bottom

图8:



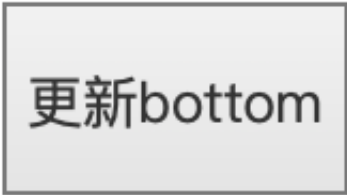
```
graph TD; A[替换bottom] --> B[更新bottom];
```

替换bottom

新的添加大于bottom，那么替换bottom。

## 更新bottom

图9:



```
graph TD; A[更新bottom];
```

更新bottom

替换了bottom后，需要调整bottom，因为新的添加只跟bottom作了比较，它有可能比其他的域值更大。

## 例子

图10:

```
Document doc ;
// 0
doc = new Document();
doc.add(new BinaryDocValuesField(fieldName, new BytesRef( text: "c")));
doc.add(new TextField( name: "superStar", value: "Andy", Field.Store.YES));
indexWriter.addDocument(doc);
// 1
doc = new Document();
doc.add(new BinaryDocValuesField(fieldName, new BytesRef( text: "b")));
doc.add(new TextField( name: "superStar", value: "Eason", Field.Store.YES));
indexWriter.addDocument(doc);
// 2
doc = new Document();
doc.add(new BinaryDocValuesField(fieldName, new BytesRef( text: "d")));
doc.add(new TextField( name: "superStar", value: "Jay", Field.Store.YES));
indexWriter.addDocument(doc);
// 3
doc = new Document();
doc.add(new BinaryDocValuesField(fieldName, new BytesRef( text: "e")));
doc.add(new TextField( name: "superStar", value: "Jolin", Field.Store.YES));
indexWriter.addDocument(doc);
// 4
doc = new Document();
doc.add(new BinaryDocValuesField(fieldName, new BytesRef( text: "a")));
doc.add(new TextField( name: "superStar", value: "KUN", Field.Store.YES));
indexWriter.addDocument(doc);
indexWriter.commit();

IndexReader reader = DirectoryReader.open(indexWriter);
IndexSearcher searcher = new IndexSearcher(reader);
```

图11:

```
Sort sort = new Sort(new SortField(fieldName, SortField.Type.STRING_VAL));
TopDocs docs = searcher.search(new MatchAllDocsQuery(), n: 3, sort);
```

文档会按照从小到大的传到collect(int doc)方法中，所以域值的出现顺序即  
"c"、"b"、"d"、"e"、"a"。

## 处理文档号0、1、2

本例子中TopN的N值为3，所以添加这三篇文档时，直接将对应的域值添加到value[]数组即可。添加结束后，队列已满，故需要设置bottom值。

图12:

bottom = d

value[]数组

c	b	d
0	1	2

### 处理文档号3

文档号3对应的域值为"e"，它比bottom值"d"小，所以直接跳过。

### 处理文档号4

文档号4对应的域值为"a"，它比bottom值"d"大，所以替换bottom，然后再更新bottom。

图13：

bottom = d

value[]数组

c	b	d
0	1	2



bottom = c

value[]数组

c	b	a
0	1	2

## 结语

本篇文章介绍了FileComparator的排序过程，出于仅原理的理解，只介绍了基本的流程，其细节的部分比如说，如果排序对象为STRING，并且在搜索阶段存在多个IndexReader时，当IndexSearcher切换IndexReader时，还要考虑在不同的IndexReader中同一个域值可能有不同的ord值的情况。完整的过程大家看源码吧，然后就是由于源码比较简单，所以我并没有添加源码注释~

