Package org.apache.lucene.search.vectorhighlight

Another highlighter implementation based on term vectors.

See: Description

Interface Summary

Interface	Description
BoundaryScanner	Finds fragment boundaries: pluggable into `BaseFragmentsBuilder`
FragListBuilder	FragListBuilder is an interface for FieldFragList builder classes.
FragmentsBuilder	`FragmentsBuilder` is an interface for fragments (snippets) builder classes.

Class Summary

Class	Description
BaseFragListBuilder	A abstract implementation of `FragListBuilder`.
BaseFragmentsBuilder	Base FragmentsBuilder implementation that supports colored pre/post tags and multivalued fields.
BreakIteratorBoundaryScanner	A `BoundaryScanner` implementation that uses `BreakIterator` to find boundaries in the text.
FastVectorHighlighter	Another highlighter implementation.
FieldFragList	FieldFragList has a list of "frag info" that is used by FragmentsBuilder class to create fragments (snippets).
FieldFragList.WeightedFragInfo	List of term offsets + weight for a frag info
FieldFragList.WeightedFragInfo.SubInfo	Represents the list of term offsets for some text
FieldPhraseList	FieldPhraseList has a list of WeightedPhraseInfo that is used by FragListBuilder to create a FieldFragList object.
FieldPhraseList.WeightedPhraseInfo	Represents the list of term offsets and boost for some text
FieldPhraseList.WeightedPhraseInfo.Toffs	Term offsets (start + end)
FieldQuery	FieldQuery breaks down query object into terms/phrases and keeps them in a QueryPhraseMap structure.
FieldQuery.QueryPhraseMap	Internal structure of a query for highlighting: represents a nested query structure
FieldTermStack	`FieldTermStack` is a stack that keeps query terms in the specified field of the document to be highlighted.
FieldTermStack.TermInfo	Single term with its position/offsets in the document and IDF weight.
ScoreOrderFragmentsBuilder	An implementation of FragmentsBuilder that outputs score-order fragments.
ScoreOrderFragmentsBuilder.ScoreComparator	Comparator for `FieldFragList.WeightedFragInfo` by boost, breaking ties by offset.
SimpleBoundaryScanner	Simple boundary scanner implementation that divides fragments based on a set of separator characters.
SimpleFieldFragList	A simple implementation of `FieldFragList`.
SimpleFragListBuilder	A simple implementation of `FragListBuilder`.
SimpleFragmentsBuilder	A simple implementation of FragmentsBuilder.
SingleFragListBuilder	An implementation class of `FragListBuilder` that generates one `FieldFragList.WeightedFragInfo` object.
WeightedFieldFragList	A weighted implementation of `FieldFragList`.

WeightedFragListBuilder	A weighted implementation of `FragListBuilder`.

Package org.apache.lucene.search.vectorhighlight Description

Another highlighter implementation based on term vectors.

Features

- fast for large docs
- support N-gram fields
- support phrase-unit highlighting with slops
- support multi-term (includes wildcard, range, regexp, etc) queries
- highlight fields need to be stored with Positions and Offsets
- take into account query boost and/or IDF-weight to score fragments
- support colored highlight tags
- pluggable FragListBuilder / FieldFragList
- pluggable FragmentsBuilder

Algorithm

To explain the algorithm, let's use the following sample text (to be highlighted) and user query:

Sample Text	Lucene is a search engine library.
User Query	Lucene^2 OR "search library"~1

The user query is a BooleanQuery that consists of TermQuery("Lucene") with boost of 2 and PhraseQuery("search library") with slop of 1.

For your convenience, here is the offsets and positions info of the sample text.

```
+--------+-----------------------------------+
|        |          1111111111222222222233333|
|  offset|01234567890123456789012345678901234|
+--------+-----------------------------------+
|document|Lucene is a search engine library. |
+--------*-----------------------------------+
|position|0      1  2 3      4      5         |
+--------*-----------------------------------+
```

Step 1.

In Step 1, Fast Vector Highlighter generates `FieldQuery.QueryPhraseMap` from the user query. `QueryPhraseMap` consists of the following members:

```
public class QueryPhraseMap {
  boolean terminal;
  int slop;   // valid if terminal == true and phraseHighlight == true
  float boost;  // valid if terminal == true
  Map<String, QueryPhraseMap> subMap;
}
```

`QueryPhraseMap` has subMap. The key of the subMap is a term text in the user query and the value is a subsequent `QueryPhraseMap`. If the query is a term (not phrase), then the subsequent `QueryPhraseMap` is marked as terminal. If the query is a phrase, then the subsequent `QueryPhraseMap` is not a terminal and it has the next term text in the phrase.

From the sample user query, the following `QueryPhraseMap` will be generated:

```
QueryPhraseMap
+--------+-+  +-------+-+
|"Lucene"|o+->|boost=2|*|  * : terminal
+--------+-+  +-------+-+

+--------+-+  +---------+-+  +-------+------+-+
|"search"|o+->|"library"|o+->|boost=1|slop=1|*|
+--------+-+  +---------+-+  +-------+------+-+
```

Step 2.

In Step 2, Fast Vector Highlighter generates `FieldTermStack`. Fast Vector Highlighter uses term vector data (must be stored `FieldType.setStoreTermVectorOffsets(boolean)` and `FieldType.setStoreTermVectorPositions(boolean)`) to generate it. `FieldTermStack` keeps the terms in the user query. Therefore, in this sample case, Fast Vector Highlighter generates the following `FieldTermStack`:

```
FieldTermStack
+------------------+
|"Lucene"(0,6,0)   |
+------------------+
|"search"(12,18,3) |
+------------------+
|"library"(26,33,5)|
+------------------+
where : "termText"(startOffset,endOffset,position)
```

Step 3.

In Step 3, Fast Vector Highlighter generates `FieldPhraseList` by reference to `QueryPhraseMap` and `FieldTermStack`.

```
FieldPhraseList
+----------------+-----------------+---+
|"Lucene"        |[(0,6)]          |w=2|
+----------------+-----------------+---+
|"search library"|[(12,18),(26,33)]|w=1|
+----------------+-----------------+---+
```

The type of each entry is `WeightedPhraseInfo` that consists of an array of terms offsets and weight.

Step 4.

In Step 4, Fast Vector Highlighter creates `FieldFragList` by reference to `FieldPhraseList`. In this sample case, the following `FieldFragList` will be generated:

```
FieldFragList
+---------------------------------+
|"Lucene"[(0,6)]                  |
|"search library"[(12,18),(26,33)]|
|totalBoost=3                     |
+---------------------------------+
```

The calculation for each `FieldFragList.WeightedFragInfo.totalBoost` (weight) depends on the implementation of `FieldFragList.add( ... )`:

```
  public void add( int startOffset, int endOffset, List<WeightedPhraseInfo> phraseInfoList ) {
    float totalBoost = 0;
    List<SubInfo> subInfos = new ArrayList<SubInfo>();
    for( WeightedPhraseInfo phraseInfo : phraseInfoList ){
      subInfos.add( new SubInfo( phraseInfo.getText(), phraseInfo.getTermsOffsets(), phraseInfo.getSeqnum() ) );
      totalBoost += phraseInfo.getBoost();
    }
    getFragInfos().add( new WeightedFragInfo( startOffset, endOffset, subInfos, totalBoost ) );
  }
```

The used implementation of `FieldFragList` is noted in `BaseFragListBuilder.createFieldFragList( ... )`:

```
  public FieldFragList createFieldFragList( FieldPhraseList fieldPhraseList, int fragCharSize ){
    return createFieldFragList( fieldPhraseList, new SimpleFieldFragList( fragCharSize ), fragCharSize );
  }
```

Currently there are basically to approaches available:

- `SimpleFragListBuilder using SimpleFieldFragList`: *sum-of-boosts*-approach. The totalBoost is calculated by summarizing the query-boosts per term. Per default a term is boosted by 1.0
- `WeightedFragListBuilder using WeightedFieldFragList`: *sum-of-distinct-weights*-approach. The totalBoost is calculated by summarizing the IDF-weights of distinct terms.

Comparison of the two approaches:

query = das alte testament (The Old Testament)

Terms in fragment	sum-of-distinct-weights	sum-of-boosts
das alte testament	5.339621	3.0
das alte testament	5.339621	3.0
das testament alte	5.339621	3.0
das alte testament	5.339621	3.0
das testament	2.9455688	2.0
das alte	2.4759595	2.0
das das das das	1.5015357	4.0
das das das	1.3003681	3.0
das das	1.061746	2.0
alte	1.0	1.0
alte	1.0	1.0
das	0.7507678	1.0
das	0.7507678	1.0
das	0.7507678	1.0
das	0.7507678	1.0
das	0.7507678	1.0

Step 5.

In Step 5, by using `FieldFragList` and the field stored data, Fast Vector Highlighter creates highlighted snippets!

Copyright © 2000-2017 Apache Software Foundation. All Rights Reserved.