[Lucene增强功能：Payload的应用](http://blog.csdn.net/shirdrn/article/details/6785481)

Open Source（38） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录[(?)](http://blog.csdn.net/shirdrn/article/details/6785481)[[+]](http://blog.csdn.net/shirdrn/article/details/6785481" \o "展开)

有关Lucene的Payload的相关内容，可以参考如下链接，介绍的非常详细，值得参考：

<http://www.ibm.com/developerworks/cn/opensource/os-cn-lucene-pl/>  
<http://www.lucidimagination.com/blog/2009/08/05/getting-started-with-payloads/>

例如，有这样的一个需求：

现在有两篇文档内容非常相似，如下所示：

**[html]** [view plain](http://blog.csdn.net/shirdrn/article/details/6785481) [copy](http://blog.csdn.net/shirdrn/article/details/6785481)

1. 文档1：egg tomato potato bread
2. 文档2：egg book potato bread

现在我想要查询食物（foods），而且是查询关键词是egg，如何能够区别出上面两个文档哪一个更是我想要的？

可以看到上面两篇文档，文档1中描述的各项都是食物，而文档2中的book不是食物，基于上述需求，应该是文档1比文档2更相关，在查询结果中，文档1排名应该更靠前。通过上面  
<http://www.lucidimagination.com/blog/2009/08/05/getting-started-with-payloads/>中给出的方法，可以在文档中，对给定词出现在文档的出现的权重信息（egg在文档1与文档中，以foods来衡量，文档1更相关），可以在索引之前处理一下，为egg增加payload信息，例如：

**[html]** [view plain](http://blog.csdn.net/shirdrn/article/details/6785481) [copy](http://blog.csdn.net/shirdrn/article/details/6785481)

1. 文档1：egg|0.984 tomato potato bread
2. 文档2：egg|0.356 book potato bread

然后再进行索引，通过Lucene提供的PayloadTermQuery就能够分辨出上述egg这个Term的不同。在Lucene中，实际上是将我们存储的Payload数据，如上述“|”分隔后面的数字，乘到了tf上，然后在进行权重的计算。

下面，我们再看一下，增加一个Field来存储Payload数据，而源文档不需要进行修改，或者，我们可以在索引之前对文档进行一个处理，例如分类，通过分类可以给不同的文档所属类别的不同程度，计算一个Payload数值。

为了能够使用存储的Payload数据信息，结合上面提出的实例，我们需要按照如下步骤去做：

**第一，待索引数据处理**

例如，增加category这个Field存储类别信息，content这个Field存储上面的内容：

**[html]** [view plain](http://blog.csdn.net/shirdrn/article/details/6785481) [copy](http://blog.csdn.net/shirdrn/article/details/6785481)

1. 文档1：
2. new Field("category", "foods|0.984 shopping|0.503", Field.Store.YES, Field.Index.ANALYZED)
3. new Field("content", "egg tomato potato bread", Field.Store.YES, Field.Index.ANALYZED)
4. 文档2：
5. new Field("category", "foods|0.356 shopping|0.791", Field.Store.YES, Field.Index.ANALYZED)
6. new Field("content", "egg book potato bread", Field.Store.YES, Field.Index.ANALYZED)

**第二，实现解析Payload数据的Analyzer**

由于Payload信息存储在category这个Field中，多个类别之间使用空格分隔，每个类别内容是以“|”分隔的，所以我们的Analyzer就要能够解析它。Lucene提供了DelimitedPayloadTokenFilter，能够处理具有分隔符的情况。我们的实现如下所示：

**[java]** [view plain](http://blog.csdn.net/shirdrn/article/details/6785481) [copy](http://blog.csdn.net/shirdrn/article/details/6785481)

1. **package** org.shirdrn.lucene.query.payloadquery;
3. **import** java.io.Reader;
5. **import** org.apache.lucene.analysis.Analyzer;
6. **import** org.apache.lucene.analysis.TokenStream;
7. **import** org.apache.lucene.analysis.WhitespaceTokenizer;
8. **import** org.apache.lucene.analysis.payloads.DelimitedPayloadTokenFilter;
9. **import** org.apache.lucene.analysis.payloads.PayloadEncoder;
11. **public** **class** PayloadAnalyzer **extends** Analyzer {
12. **private** PayloadEncoder encoder;
14. PayloadAnalyzer(PayloadEncoder encoder) {
15. **this**.encoder = encoder;
16. }
18. @SuppressWarnings("deprecation")
19. **public** TokenStream tokenStream(String fieldName, Reader reader) {
20. TokenStream result = **new** WhitespaceTokenizer(reader); // 用来解析空格分隔的各个类别
21. result = **new** DelimitedPayloadTokenFilter(result, '|', encoder); // 在上面分词的基础上，在进行Payload数据解析
22. **return** result;
23. }
24. }

**第三， 实现Similarity计算得分**

Lucene中Similarity类中提供了scorePayload方法，用于计算Payload值来对文档贡献得分，我们重写了该方法，实现如下所示：

**[java]** [view plain](http://blog.csdn.net/shirdrn/article/details/6785481) [copy](http://blog.csdn.net/shirdrn/article/details/6785481)

1. **package** org.shirdrn.lucene.query.payloadquery;
3. **import** org.apache.lucene.analysis.payloads.PayloadHelper;
4. **import** org.apache.lucene.search.DefaultSimilarity;

7. **public** **class** PayloadSimilarity **extends** DefaultSimilarity {
9. **private** **static** **final** **long** serialVersionUID = 1L;
11. @Override
12. **public** **float** scorePayload(**int** docId, String fieldName, **int** start, **int** end,
13. **byte**[] payload, **int** offset, **int** length) {
14. **return** PayloadHelper.decodeFloat(payload, offset);
15. }
17. }

通过使用PayloadHelper这个工具类可以获取到Payload值，然后在计算文档得分的时候起到作用。

**第四，创建索引**

在创建索引的时候，需要使用到我们上面实现的Analyzer和Similarity，代码如下所示：

**[java]** [view plain](http://blog.csdn.net/shirdrn/article/details/6785481) [copy](http://blog.csdn.net/shirdrn/article/details/6785481)

1. **package** org.shirdrn.lucene.query.payloadquery;
3. **import** java.io.File;
4. **import** java.io.IOException;
6. **import** org.apache.lucene.analysis.Analyzer;
7. **import** org.apache.lucene.analysis.payloads.FloatEncoder;
8. **import** org.apache.lucene.document.Document;
9. **import** org.apache.lucene.document.Field;
10. **import** org.apache.lucene.index.CorruptIndexException;
11. **import** org.apache.lucene.index.IndexWriter;
12. **import** org.apache.lucene.index.IndexWriterConfig;
13. **import** org.apache.lucene.index.IndexWriterConfig.OpenMode;
14. **import** org.apache.lucene.search.Similarity;
15. **import** org.apache.lucene.store.FSDirectory;
16. **import** org.apache.lucene.store.LockObtainFailedException;
17. **import** org.apache.lucene.util.Version;
19. **public** **class** PayloadIndexing {
21. **private** IndexWriter indexWriter = **null**;
22. **private** **final** Analyzer analyzer = **new** PayloadAnalyzer(**new** FloatEncoder()); // 使用PayloadAnalyzer，并指定Encoder
23. **private** **final** Similarity similarity = **new** PayloadSimilarity(); // 实例化一个PayloadSimilarity
24. **private** IndexWriterConfig config = **null**;
26. **public** PayloadIndexing(String indexPath) **throws** CorruptIndexException, LockObtainFailedException, IOException {
27. File indexFile = **new** File(indexPath);
28. config = **new** IndexWriterConfig(Version.LUCENE\_31, analyzer);
29. config.setOpenMode(OpenMode.CREATE).setSimilarity(similarity); // 设置计算得分的Similarity
30. indexWriter = **new** IndexWriter(FSDirectory.open(indexFile), config);
31. }
33. **public** **void** index() **throws** CorruptIndexException, IOException {
34. Document doc1 = **new** Document();
35. doc1.add(**new** Field("category", "foods|0.984 shopping|0.503", Field.Store.YES, Field.Index.ANALYZED));
36. doc1.add(**new** Field("content", "egg tomato potato bread", Field.Store.YES, Field.Index.ANALYZED));
37. indexWriter.addDocument(doc1);
39. Document doc2 = **new** Document();
40. doc2.add(**new** Field("category", "foods|0.356 shopping|0.791", Field.Store.YES, Field.Index.ANALYZED));
41. doc2.add(**new** Field("content", "egg book potato bread", Field.Store.YES, Field.Index.ANALYZED));
42. indexWriter.addDocument(doc2);
44. indexWriter.close();
45. }
47. **public** **static** **void** main(String[] args) **throws** CorruptIndexException, IOException {
48. **new** PayloadIndexing("E:\\index").index();
49. }
50. }

**第五，查询**

查询的时候，我们可以构造PayloadTermQuery来进行查询。代码如下所示：

**[java]** [view plain](http://blog.csdn.net/shirdrn/article/details/6785481) [copy](http://blog.csdn.net/shirdrn/article/details/6785481)

1. **package** org.shirdrn.lucene.query.payloadquery;
3. **import** java.io.File;
4. **import** java.io.IOException;
6. **import** org.apache.lucene.document.Document;
7. **import** org.apache.lucene.index.CorruptIndexException;
8. **import** org.apache.lucene.index.IndexReader;
9. **import** org.apache.lucene.index.Term;
10. **import** org.apache.lucene.queryParser.ParseException;
11. **import** org.apache.lucene.search.BooleanQuery;
12. **import** org.apache.lucene.search.Explanation;
13. **import** org.apache.lucene.search.IndexSearcher;
14. **import** org.apache.lucene.search.ScoreDoc;
15. **import** org.apache.lucene.search.TopScoreDocCollector;
16. **import** org.apache.lucene.search.BooleanClause.Occur;
17. **import** org.apache.lucene.search.payloads.AveragePayloadFunction;
18. **import** org.apache.lucene.search.payloads.PayloadTermQuery;
19. **import** org.apache.lucene.store.NIOFSDirectory;
21. **public** **class** PayloadSearching {
23. **private** IndexReader indexReader;
24. **private** IndexSearcher searcher;
26. **public** PayloadSearching(String indexPath) **throws** CorruptIndexException, IOException {
27. indexReader = IndexReader.open(NIOFSDirectory.open(**new** File(indexPath)), **true**);
28. searcher = **new** IndexSearcher(indexReader);
29. searcher.setSimilarity(**new** PayloadSimilarity()); // 设置自定义的PayloadSimilarity
30. }
32. **public** ScoreDoc[] search(String qsr) **throws** ParseException, IOException {
33. **int** hitsPerPage = 10;
34. BooleanQuery bq = **new** BooleanQuery();
35. **for**(String q : qsr.split(" ")) {
36. bq.add(createPayloadTermQuery(q), Occur.MUST);
37. }
38. TopScoreDocCollector collector = TopScoreDocCollector.create(5 \* hitsPerPage, **true**);
39. searcher.search(bq, collector);
40. ScoreDoc[] hits = collector.topDocs().scoreDocs;
41. **for** (**int** i = 0; i < hits.length; i++) {
42. **int** docId = hits[i].doc; // 文档编号
43. Explanation  explanation  = searcher.explain(bq, docId);
44. System.out.println(explanation.toString());
45. }
46. **return** hits;
47. }
49. **public** **void** display(ScoreDoc[] hits, **int** start, **int** end) **throws** CorruptIndexException, IOException {
50. end = Math.min(hits.length, end);
51. **for** (**int** i = start; i < end; i++) {
52. Document doc = searcher.doc(hits[i].doc);
53. **int** docId = hits[i].doc; // 文档编号
54. **float** score = hits[i].score; // 文档得分
55. System.out.println(docId + "\t" + score + "\t" + doc + "\t");
56. }
57. }
59. **public** **void** close() **throws** IOException {
60. searcher.close();
61. indexReader.close();
62. }
64. **private** PayloadTermQuery createPayloadTermQuery(String item) {
65. PayloadTermQuery ptq = **null**;
66. **if**(item.indexOf("^")!=-1) {
67. String[] a = item.split("\\^");
68. String field = a[0].split(":")[0];
69. String token = a[0].split(":")[1];
70. ptq = **new** PayloadTermQuery(**new** Term(field, token), **new** AveragePayloadFunction());
71. ptq.setBoost(Float.parseFloat(a[1].trim()));
72. } **else** {
73. String field = item.split(":")[0];
74. String token = item.split(":")[1];
75. ptq = **new** PayloadTermQuery(**new** Term(field, token), **new** AveragePayloadFunction());
76. }
77. **return** ptq;
78. }
80. **public** **static** **void** main(String[] args) **throws** ParseException, IOException {
81. **int** start = 0, end = 10;
82. //      String queries = "category:foods^123.0 content:bread^987.0";
83. String queries = "category:foods content:egg";
84. PayloadSearching payloadSearcher = **new** PayloadSearching("E:\\index");
85. payloadSearcher.display(payloadSearcher.search(queries), start, end);
86. payloadSearcher.close();
87. }
89. }

我们可以看到查询结果，两个文档的相关度排序：

**[html]** [view plain](http://blog.csdn.net/shirdrn/article/details/6785481) [copy](http://blog.csdn.net/shirdrn/article/details/6785481)

1. 0   0.3314532   Document**<stored**,indexed,tokenized**<category:foods**|0.984 shopping|0.503**>** stored,indexed,tokenized**<content:egg** tomato potato bread**>>**
2. 1   0.21477573  Document**<stored**,indexed,tokenized**<category:foods**|0.356 shopping|0.791**>** stored,indexed,tokenized**<content:egg** book potato bread**>>**

通过输出计算得分的解释信息，如下所示：

**[html]** [view plain](http://blog.csdn.net/shirdrn/article/details/6785481) [copy](http://blog.csdn.net/shirdrn/article/details/6785481)

1. 0.3314532 = (MATCH) sum of:
2. 0.18281947 = (MATCH) weight(category:foods in 0), product of:
3. 0.70710677 = queryWeight(category:foods), product of:
4. 0.5945349 = idf(category:  foods=2)
5. 1.1893445 = queryNorm
6. 0.2585458 = (MATCH) fieldWeight(category:foods in 0), product of:
7. 0.6957931 = (MATCH) btq, product of:
8. 0.70710677 = tf(phraseFreq=0.5)
9. 0.984 = scorePayload(...)
10. 0.5945349 = idf(category:  foods=2)
11. 0.625 = fieldNorm(field=category, doc=0)
12. 0.14863372 = (MATCH) weight(content:egg in 0), product of:
13. 0.70710677 = queryWeight(content:egg), product of:
14. 0.5945349 = idf(content:  egg=2)
15. 1.1893445 = queryNorm
16. 0.21019982 = (MATCH) fieldWeight(content:egg in 0), product of:
17. 0.70710677 = (MATCH) btq, product of:
18. 0.70710677 = tf(phraseFreq=0.5)
19. 1.0 = scorePayload(...)
20. 0.5945349 = idf(content:  egg=2)
21. 0.5 = fieldNorm(field=content, doc=0)
23. 0.21477571 = (MATCH) sum of:
24. 0.066142 = (MATCH) weight(category:foods in 1), product of:
25. 0.70710677 = queryWeight(category:foods), product of:
26. 0.5945349 = idf(category:  foods=2)
27. 1.1893445 = queryNorm
28. 0.09353892 = (MATCH) fieldWeight(category:foods in 1), product of:
29. 0.25173002 = (MATCH) btq, product of:
30. 0.70710677 = tf(phraseFreq=0.5)
31. 0.356 = scorePayload(...)
32. 0.5945349 = idf(category:  foods=2)
33. 0.625 = fieldNorm(field=category, doc=1)
34. 0.14863372 = (MATCH) weight(content:egg in 1), product of:
35. 0.70710677 = queryWeight(content:egg), product of:
36. 0.5945349 = idf(content:  egg=2)
37. 1.1893445 = queryNorm
38. 0.21019982 = (MATCH) fieldWeight(content:egg in 1), product of:
39. 0.70710677 = (MATCH) btq, product of:
40. 0.70710677 = tf(phraseFreq=0.5)
41. 1.0 = scorePayload(...)
42. 0.5945349 = idf(content:  egg=2)
43. 0.5 = fieldNorm(field=content, doc=1)

我们可以看到，除了在计算category权重的时候，tf上乘了一个Payload值以外，其他的都相同，也就是说，我们预期使用的Payload为文档（ID=0）贡献了得分，排名靠前了。否则，如果不使用Payload的话，查询结果中两个文档的得分是相同的（可以模拟设置他们的Payload值相同，测试一下看看