

原 LIRe 源代码分析 6：检索（ImageSearcher）[以颜色布局为例]

2013年11月02日 20:43:39 阅读数：4715

=====

LIRe源代码分析系列文章列表：

[LIRe 源代码分析 1：整体结构](#)

[LIRe 源代码分析 2：基本接口（DocumentBuilder）](#)

[LIRe 源代码分析 3：基本接口（ImageSearcher）](#)

[LIRe 源代码分析 4：建立索引（DocumentBuilder）\[以颜色布局为例\]](#)

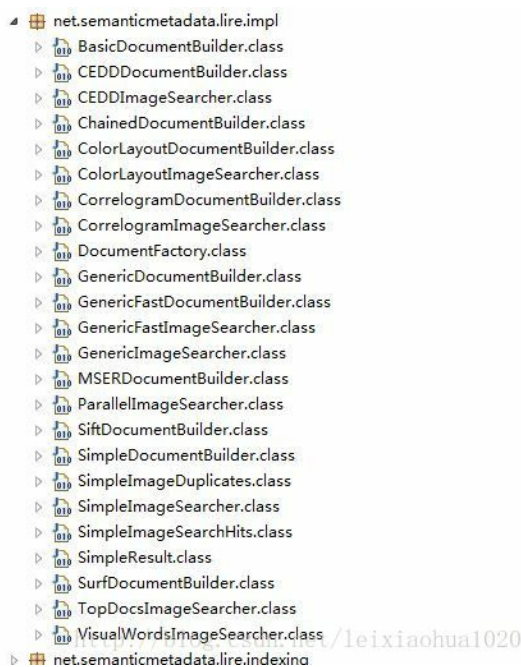
[LIRe 源代码分析 5：提取特征向量\[以颜色布局为例\]](#)

[LIRe 源代码分析 6：检索（ImageSearcher）\[以颜色布局为例\]](#)

[LIRe 源代码分析 7：算法类\[以颜色布局为例\]](#)

=====

前几篇文章介绍了LIRe 的基本接口，以及建立索引的过程。现在来看一看它的检索部分（ImageSearcher）。不同的方法的检索功能的类各不相同，它们都位于“net.semanticmetadata.lire.impl”中，如下图所示：



在这里仅分析一个比较有代表性的：颜色布局。前文已经分析过ColorLayoutDocumentBuilder，在这里我们分析一下ColorLayoutImageSearcher。源代码如下：

```
[java]
1.  /*
2.   * This file is part of the LIRe project: http://www.semanticmetadata.net/lire
3.   * LIRe is free software; you can redistribute it and/or modify
4.   * it under the terms of the GNU General Public License as published by
5.   * the Free Software Foundation; either version 2 of the License, or
6.   * (at your option) any later version.
7.   *
8.   * LIRe is distributed in the hope that it will be useful,
9.   * but WITHOUT ANY WARRANTY; without even the implied warranty of
10.  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
11.  * GNU General Public License for more details.
12.  *
13.  * You should have received a copy of the GNU General Public License
14.  * along with LIRe; if not, write to the Free Software
15.  * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
16.  *
17.  * We kindly ask you to refer the following paper in any publication mentioning Lire:
```

```

18.  *
19.  * Lux Mathias, Savvas A. Chatzichristofis. Lire: Lucene Image Retrieval 鈥?
20.  * An Extensible Java CBIR Library. In proceedings of the 16th ACM International
21.  * Conference on Multimedia, pp. 1085-1088, Vancouver, Canada, 2008
22.  *
23.  * http://doi.acm.org/10.1145/1459359.1459577
24.  *
25.  * Copyright statement:
26.  * -----
27.  * (c) 2002-2011 by Mathias Lux (mathias@juggle.at)
28.  * http://www.semanticmetadata.net/lire
29.  */
30. package net.semanticmetadata.lire.impl;
31.
32. import net.semanticmetadata.lire.DocumentBuilder;
33. import net.semanticmetadata.lire.ImageDuplicates;
34. import net.semanticmetadata.lire.ImageSearchHits;
35. import net.semanticmetadata.lire.imageanalysis.ColorLayout;
36. import net.semanticmetadata.lire.imageanalysis.LireFeature;
37. import org.apache.lucene.document.Document;
38. import org.apache.lucene.index.IndexReader;
39.
40. import java.io.FileNotFoundException;
41. import java.io.IOException;
42. import java.util.HashMap;
43. import java.util.LinkedList;
44. import java.util.List;
45. import java.util.logging.Level;
46.
47. /**
48.  * Provides a faster way of searching based on byte arrays instead of Strings. The method
49.  * {@link net.semanticmetadata.lire.imageanalysis.ColorLayout#getBytesRepresentation()} is used
50.  * to generate the signature of the descriptor much faster. First tests have shown that this
51.  * implementation is up to 4 times faster than the implementation based on strings
52.  * (for 120,000 images)
53.  * <p>
54.  * User: Mathias Lux, mathias@juggle.at
55.  * Date: 30.06 2011
56.  */
57. public class ColorLayoutImageSearcher extends GenericImageSearcher {
58.     public ColorLayoutImageSearcher(int maxHits) {
59         super(maxHits, ColorLayout.class, DocumentBuilder.FIELD_NAME_COLORLAYOUT_FAST);
60     }
61.
62.     protected float getDistance(Document d, LireFeature lireFeature) {
63         float distance = 0f;
64         ColorLayout lf;
65         try {
66             lf = (ColorLayout) descriptorClass.newInstance();
67             byte[] cls = d.getBinaryValue(fieldName);
68             if (cls != null && cls.length > 0) {
69                 lf.setBytesRepresentation(cls);
70                 distance = lireFeature.getDistance(lf);
71             } else {
72                 logger.warning("No feature stored in this document ...");
73             }
74         } catch (InstantiationException e) {
75             logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
76         } catch (IllegalAccessException e) {
77             logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
78         }
79.
80         return distance;
81     }
82.
83.     public ImageSearchHits search(Document doc, IndexReader reader) throws IOException {
84         SimpleImageSearchHits searchHits = null;
85         try {
86             ColorLayout lireFeature = (ColorLayout) descriptorClass.newInstance();
87.
88             byte[] cls = doc.getBinaryValue(fieldName);
89             if (cls != null && cls.length > 0)
90                 lireFeature.setBytesRepresentation(cls);
91             float maxDistance = findSimilar(reader, lireFeature);
92.
93             searchHits = new SimpleImageSearchHits(this.docs, maxDistance);
94         } catch (InstantiationException e) {
95             logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
96         } catch (IllegalAccessException e) {
97             logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
98         }
99         return searchHits;
100.    }
101.
102.    public ImageDuplicates findDuplicates(IndexReader reader) throws IOException {
103        // get the first document:
104        SimpleImageDuplicates simpleImageDuplicates = null;
105        try {
106            if (!IndexReader.indexExists(reader.directory()))
107                throw new FileNotFoundException("No index found at this specific location.");
108            Document doc = reader.document(0);

```

```

109.
110.     ColorLayout lireFeature = (ColorLayout) descriptorClass.newInstance();
111.     byte[] cls = doc.getBinaryValue(fieldName);
112.     if (cls != null && cls.length > 0)
113.         lireFeature.setByteArrayRepresentation(cls);
114.
115.     HashMap<Float, List<String>> duplicates = new HashMap<Float, List<String>>();
116.
117.     // find duplicates ...
118.     boolean hasDeletions = reader.hasDeletions();
119.
120.     int docs = reader.numDocs();
121.     int numDuplicates = 0;
122.     for (int i = 0; i < docs; i++) {
123.         if (hasDeletions && reader.isDeleted(i)) {
124.             continue;
125.         }
126.         Document d = reader.document(i);
127.         float distance = getDistance(d, lireFeature);
128.
129.         if (!duplicates.containsKey(distance)) {
130.             duplicates.put(distance, new LinkedList<String>());
131.         } else {
132.             numDuplicates++;
133.         }
134.         duplicates.get(distance).add(d.getFieldable(DocumentBuilder.FIELD_NAME_IDENTIFIER).stringValue());
135.     }
136.
137.     if (numDuplicates == 0) return null;
138.
139.     LinkedList<List<String>> results = new LinkedList<List<String>>();
140.     for (float f : duplicates.keySet()) {
141.         if (duplicates.get(f).size() > 1) {
142.             results.add(duplicates.get(f));
143.         }
144.     }
145.     simpleImageDuplicates = new SimpleImageDuplicates(results);
146. } catch (InstantiationException e) {
147.     logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
148. } catch (IllegalAccessException e) {
149.     logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
150. }
151. return simpleImageDuplicates;
152.
153. }
154. }

```

源代码里面重要的函数有3个：

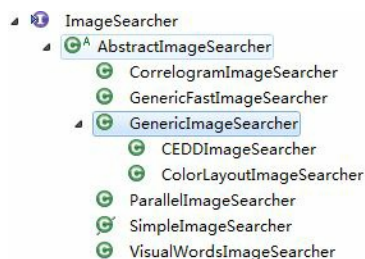
float getDistance(Document d, LireFeature lireFeature)：

ImageSearchHits search(Document doc, IndexReader reader)：检索。最核心函数。

ImageDuplicates findDuplicates(IndexReader reader)：目前还没研究。

在这里忽然发现了一个问题：这里竟然只有一个Search()？！应该是有参数不同的3个Search()才对啊.....

经过研究后发现，ColorLayoutImageSearcher继承了一个类——GenericImageSearcher，而不是继承AbstractImageSearcher。Search()方法的实现是在GenericImageSearcher中实现的。看来这个ColorLayoutImageSearcher还挺特殊的啊.....



看一下GenericImageSearcher的源代码：

```

1. package net.semanticmetadata.lire.impl;
2.
3. import net.semanticmetadata.lire.AbstractImageSearcher;
4. import net.semanticmetadata.lire.DocumentBuilder;
5. import net.semanticmetadata.lire.ImageDuplicates;
6. import net.semanticmetadata.lire.ImageSearchHits;
7. import net.semanticmetadata.lire.imageanalysis.LireFeature;
8. import net.semanticmetadata.lire.utils.ImageUtils;
9. import org.apache.lucene.document.Document;
10. import org.apache.lucene.index.IndexReader;

```

```

11.
12. import java.awt.image.BufferedImage;
13. import java.io.FileNotFoundException;
14. import java.io.IOException;
15. import java.util.HashMap;
16. import java.util.LinkedList;
17. import java.util.List;
18. import java.util.TreeSet;
19. import java.util.logging.Level;
20. import java.util.logging.Logger;
21.
22. /**
23.  * This file is part of the Caliph and Emir project: http://www.SemanticMetadata.net
24.  * <br>Date: 01.02.2006
25.  * <br>Time: 00:17:02
26.  *
27.  * @author Mathias Lux, mathias@juggle.at
28.  */
29. public class GenericImageSearcher extends AbstractImageSearcher {
30.     protected Logger logger = Logger.getLogger(getClass().getName());
31.     Class<?> descriptorClass;
32.     String fieldName;
33.
34.     private int maxHits = 10;
35.     protected TreeSet<SimpleResult> docs;
36.
37.     public GenericImageSearcher(int maxHits, Class<?> descriptorClass, String fieldName) {
38.         this.maxHits = maxHits;
39.         docs = new TreeSet<SimpleResult>();
40.         this.descriptorClass = descriptorClass;
41.         this.fieldName = fieldName;
42.     }
43.
44.     public ImageSearchHits search(BufferedImage image, IndexReader reader) throws IOException {
45.         logger.finer("Starting extraction.");
46.         LireFeature lireFeature = null;
47.         SimpleImageSearchHits searchHits = null;
48.         try {
49.             lireFeature = (LireFeature) descriptorClass.newInstance();
50.             // Scaling image is especially with the correlogram features very important!
51.             BufferedImage bimg = image;
52.             if (Math.max(image.getHeight(), image.getWidth()) > GenericDocumentBuilder.MAX_IMAGE_DIMENSION) {
53.                 bimg = ImageUtils.scaleImage(image, GenericDocumentBuilder.MAX_IMAGE_DIMENSION);
54.             }
55.             lireFeature.extract(bimg);
56.             logger.fine("Extraction from image finished");
57.
58.             float maxDistance = findSimilar(reader, lireFeature);
59.             searchHits = new SimpleImageSearchHits(this.docs, maxDistance);
60.         } catch (InstantiationException e) {
61.             logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
62.         } catch (IllegalAccessException e) {
63.             logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
64.         }
65.         return searchHits;
66.     }
67.
68.     /**
69.      * @param reader
70.      * @param lireFeature
71.      * @return the maximum distance found for normalizing.
72.      * @throws java.io.IOException
73.      */
74.     protected float findSimilar(IndexReader reader, LireFeature lireFeature) throws IOException {
75.         float maxDistance = -1f, overallMaxDistance = -1f;
76.         boolean hasDeletions = reader.hasDeletions();
77.
78.         // clear result set ...
79.         docs.clear();
80.
81.         int docsCount = reader.numDocs();
82.         for (int i = 0; i < docsCount; i++) {
83.             // bugfix by Roman Kern
84.             if (hasDeletions && reader.isDeleted(i)) {
85.                 continue;
86.             }
87.
88.             Document d = reader.document(i);
89.             float distance = getDistance(d, lireFeature);
90.             assert (distance >= 0);
91.             // calculate the overall max distance to normalize score afterwards
92.             if (overallMaxDistance < distance) {
93.                 overallMaxDistance = distance;
94.             }
95.             // if it is the first document:
96.             if (maxDistance < 0) {
97.                 maxDistance = distance;
98.             }
99.             // if the array is not full yet:
100.            if (this.docs.size() < maxHits) {
101.                this.docs.add(new SimpleResult(distance, d));

```

```

102.         if (distance > maxDistance) maxDistance = distance;
103.     } else if (distance < maxDistance) {
104.         // if it is nearer to the sample than at least on of the current set:
105.         // remove the last one ...
106.         this.docs.remove(this.docs.last());
107.         // add the new one ...
108.         this.docs.add(new SimpleResult(distance, d));
109.         // and set our new distance border ...
110.         maxDistance = this.docs.last().getDistance();
111.     }
112. }
113. return maxDistance;
114. }
115.
116. protected float getDistance(Document d, LireFeature lireFeature) {
117.     float distance = 0f;
118.     LireFeature lf;
119.     try {
120.         lf = (LireFeature) descriptorClass.newInstance();
121.         String[] cls = d.getValues(fieldName);
122.         if (cls != null && cls.length > 0) {
123.             lf.setStringRepresentation(cls[0]);
124.             distance = lireFeature.getDistance(lf);
125.         } else {
126.             logger.warning("No feature stored in this document!");
127.         }
128.     } catch (InstantiationException e) {
129.         logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
130.     } catch (IllegalAccessException e) {
131.         logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
132.     }
133.
134.     return distance;
135. }
136.
137. public ImageSearchHits search(Document doc, IndexReader reader) throws IOException {
138.     SimpleImageSearchHits searchHits = null;
139.     try {
140.         LireFeature lireFeature = (LireFeature) descriptorClass.newInstance();
141.
142.         String[] cls = doc.getValues(fieldName);
143.         if (cls != null && cls.length > 0)
144.             lireFeature.setStringRepresentation(cls[0]);
145.         float maxDistance = findSimilar(reader, lireFeature);
146.
147.         searchHits = new SimpleImageSearchHits(this.docs, maxDistance);
148.     } catch (InstantiationException e) {
149.         logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
150.     } catch (IllegalAccessException e) {
151.         logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
152.     }
153.     return searchHits;
154. }
155.
156. public ImageDuplicates findDuplicates(IndexReader reader) throws IOException {
157.     // get the first document:
158.     SimpleImageDuplicates simpleImageDuplicates = null;
159.     try {
160.         if (!IndexReader.indexExists(reader.directory()))
161.             throw new FileNotFoundException("No index found at this specific location.");
162.         Document doc = reader.document(0);
163.
164.         LireFeature lireFeature = (LireFeature) descriptorClass.newInstance();
165.         String[] cls = doc.getValues(fieldName);
166.         if (cls != null && cls.length > 0)
167.             lireFeature.setStringRepresentation(cls[0]);
168.
169.         HashMap<Float, List<String>> duplicates = new HashMap<Float, List<String>>();
170.
171.         // find duplicates ...
172.         boolean hasDeletions = reader.hasDeletions();
173.
174.         int docs = reader.numDocs();
175.         int numDuplicates = 0;
176.         for (int i = 0; i < docs; i++) {
177.             if (hasDeletions && reader.isDeleted(i)) {
178.                 continue;
179.             }
180.             Document d = reader.document(i);
181.             float distance = getDistance(d, lireFeature);
182.
183.             if (!duplicates.containsKey(distance)) {
184.                 duplicates.put(distance, new LinkedList<String>());
185.             } else {
186.                 numDuplicates++;
187.             }
188.             duplicates.get(distance).add(d.getFieldable(DocumentBuilder.FIELD_NAME_IDENTIFIER).stringValue());
189.         }
190.
191.         if (numDuplicates == 0) return null;
192.
193.         SimpleImageDuplicates simpleImageDuplicates = new SimpleImageDuplicates(duplicates, new LinkedList<String>());

```

```

193.         LinkedList<List<String>> results = new LinkedList<List<String>>();
194.         for (float f : duplicates.keySet()) {
195.             if (duplicates.get(f).size() > 1) {
196.                 results.add(duplicates.get(f));
197.             }
198.         }
199.         simpleImageDuplicates = new SimpleImageDuplicates(results);
200.     } catch (InstantiationException e) {
201.         logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
202.     } catch (IllegalAccessException e) {
203.         logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
204.     }
205.     return simpleImageDuplicates;
206.
207. }
208.
209. public String toString() {
210.     return "GenericSearcher using " + descriptorClass.getName();
211. }
212.
213. }

```

下面来看看GenericImageSearcher中的search(BufferedImage image, IndexReader reader)函数的步骤(注:这个函数应该是用的最多的,输入一张图片,返回相似图片的结果集)：

- 1.输入图片如果尺寸过大（大于1024），则调整尺寸。
- 2.使用extract()提取输入图片的特征值。
- 3.根据提取的特征值，使用findSimilar()查找相似的图片。
- 4.新建一个ImageSearchHits用于存储查找的结果。
- 5.返回ImageSearchHits

在这里要注意一点：

GenericImageSearcher中创建特定方法的类的时候，使用了如下形式：

```

1.  LireFeature lireFeature = (LireFeature) descriptorClass.newInstance();

```

即接口的方式，而不是直接新建一个对象的方式，形如：

```

1.  AutoColorCorrelogram acc = new AutoColorCorrelogram(CorrelogramDocumentBuilder.MAXIMUM_DISTANCE)

```

相比而言，更具有通用型。

在search()函数中，调用了函数findSimilar()。这个函数的作用是查找相似图片的，分析一下它的步骤：

- 1.使用IndexReader获取所有的记录
- 2.遍历所有的记录，和当前输入的图片进行比较，使用getDistance()函数
- 3.获取maxDistance并返回

在findSimilar()中，又调用了函数getDistance()，该函数调用了具体检索方法的getDistance()函数。

下面我们来看一下ColorLayout类中的getDistance()函数：

```

1.  public float getDistance(LireFeature descriptor) {
2.      if (!(descriptor instanceof ColorLayoutImpl)) return -1f;
3.      ColorLayoutImpl cl = (ColorLayoutImpl) descriptor;
4.      return (float) ColorLayoutImpl.getSimilarity(YCoeff, CbCoeff, CrCoeff, cl.YCoeff, cl.CbCoeff, cl.CrCoeff);
5.  }

```

发现其调用了ColorLayoutImpl类中的getSimilarity()函数：

```

1. public static double getSimilarity(int[] YCoeff1, int[] CbCoeff1, int[] CrCoeff1, int[] YCoeff2, int[] CbCoeff2, int[] CrCoeff2) {
2.     int numYCoeff1, numYCoeff2, CCoeff1, CCoeff2, YCoeff, CCoeff;
3.
4.     //Numbers of the Coefficients of two descriptor values.
5.     numYCoeff1 = YCoeff1.length;
6.     numYCoeff2 = YCoeff2.length;
7.     CCoeff1 = CbCoeff1.length;
8.     CCoeff2 = CbCoeff2.length;
9.
10.    //take the minimal Coeff-number
11.    YCoeff = Math.min(numYCoeff1, numYCoeff2);
12.    CCoeff = Math.min(CCoeff1, CCoeff2);
13.
14.    setWeightingValues();
15.
16.    int j;
17.    int[] sum = new int[3];
18.    int diff;
19.    sum[0] = 0;
20.
21.    for (j = 0; j < YCoeff; j++) {
22.        diff = (YCoeff1[j] - YCoeff2[j]);
23.        sum[0] += (weightMatrix[0][j] * diff * diff);
24.    }
25.
26.    sum[1] = 0;
27.    for (j = 0; j < CCoeff; j++) {
28.        diff = (CbCoeff1[j] - CbCoeff2[j]);
29.        sum[1] += (weightMatrix[1][j] * diff * diff);
30.    }
31.
32.    sum[2] = 0;
33.    for (j = 0; j < CCoeff; j++) {
34.        diff = (CrCoeff1[j] - CrCoeff2[j]);
35.        sum[2] += (weightMatrix[2][j] * diff * diff);
36.    }
37.
38.    //returns the distance between the two descriptor values
39.
40.    return Math.sqrt(sum[0] * 1.0) + Math.sqrt(sum[1] * 1.0) + Math.sqrt(sum[2] * 1.0);
41. }

```

由代码可见，getSimilarity()通过具体的算法，计算两张图片特征向量之间的相似度。

版权声明：本文为博主原创文章，未经博主允许不得转载。 <https://blog.csdn.net/leixiaohua1020/article/details/13792905>

文章标签：[lire](#) [源代码](#) [索引](#) [检索](#) [lucene](#)

个人分类：[MPEG7/图像检索](#) [LIRe](#)

所属专栏：[开源多媒体项目源代码分析](#)

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com