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]

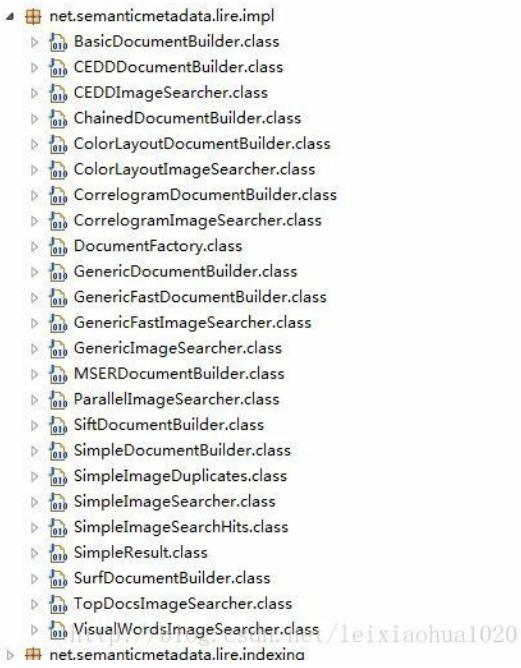
LIRe 源代码分析 5：提取特征向量[以颜色布局为例]

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

```
 *
 * Lux Mathias, Savvas A. Chatzichristofis. Lire: Lucene Image Retrieval 鈥
 * An Extensible Java CBIR Library. In proceedings of the 16th ACM International
 * Conference on Multimedia, pp. 1085-1088, Vancouver, Canada, 2008
 *
 * http://doi.acm.org/10.1145/1459359.1459577
 *
 * Copyright statement:
 * --------------------
 * (c) 2002-2011 by Mathias Lux (mathias@juggle.at)
 *     http://www.semanticmetadata.net/lire
 */
package net.semanticmetadata.lire.impl;

import net.semanticmetadata.lire.DocumentBuilder;
import net.semanticmetadata.lire.ImageDuplicates;
import net.semanticmetadata.lire.ImageSearchHits;
import net.semanticmetadata.lire.imageanalysis.ColorLayout;
import net.semanticmetadata.lire.imageanalysis.LireFeature;
import org.apache.lucene.document.Document;
import org.apache.lucene.index.IndexReader;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.logging.Level;

/**
 * Provides a faster way of searching based on byte arrays instead of Strings. The method
 * {@link net.semanticmetadata.lire.imageanalysis.ColorLayout#getByteArrayRepresentation()} is used
 * to generate the signature of the descriptor much faster. First tests have shown that this
 * implementation is up to 4 times faster than the implementation based on strings
 * (for 120,000 images)
 * <p/>
 * User: Mathias Lux, mathias@juggle.at
 * Date: 30.06 2011
 */
public class ColorLayoutImageSearcher extends GenericImageSearcher {
    public ColorLayoutImageSearcher(int maxHits) {
        super(maxHits, ColorLayout.class, DocumentBuilder.FIELD_NAME_COLORLAYOUT_FAST);
    }

    protected float getDistance(Document d, LireFeature lireFeature) {
        float distance = 0f;
        ColorLayout lf;
        try {
            lf = (ColorLayout) descriptorClass.newInstance();
            byte[] cls = d.getBinaryValue(fieldName);
            if (cls != null && cls.length > 0) {
                lf.setByteArrayRepresentation(cls);
                distance = lireFeature.getDistance(lf);
            } else {
                logger.warning("No feature stored in this document ...");
            }
        } catch (InstantiationException e) {
            logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
        } catch (IllegalAccessException e) {
            logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
        }

        return distance;
    }

    public ImageSearchHits search(Document doc, IndexReader reader) throws IOException {
        SimpleImageSearchHits searchHits = null;
        try {
            ColorLayout lireFeature = (ColorLayout) descriptorClass.newInstance();

            byte[] cls = doc.getBinaryValue(fieldName);
            if (cls != null && cls.length > 0)
                lireFeature.setByteArrayRepresentation(cls);
            float maxDistance = findSimilar(reader, lireFeature);

            searchHits = new SimpleImageSearchHits(this.docs, maxDistance);
        } catch (InstantiationException e) {
            logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
        } catch (IllegalAccessException e) {
            logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
        }
        return searchHits;
    }

    public ImageDuplicates findDuplicates(IndexReader reader) throws IOException {
        // get the first document:
        SimpleImageDuplicates simpleImageDuplicates = null;
        try {
            if (!IndexReader.indexExists(reader.directory()))
                throw new FileNotFoundException("No index found at this specific location.");
            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：

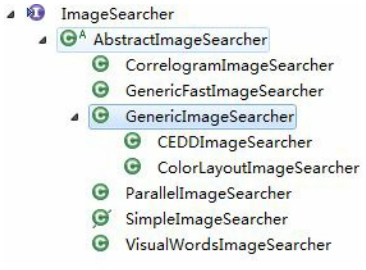
float getDistance(Document d, LireFeature lireFeature)：

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：

```
[java]
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;
```

```java

import java.awt.image.BufferedImage;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.TreeSet;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * This file is part of the Caliph and Emir project: http://www.SemanticMetadata.net
 * <br>Date: 01.02.2006
 * <br>Time: 00:17:02
 *
 * @author Mathias Lux, mathias@juggle.at
 */
public class GenericImageSearcher extends AbstractImageSearcher {
    protected Logger logger = Logger.getLogger(getClass().getName());
    Class<?> descriptorClass;
    String fieldName;

    private int maxHits = 10;
    protected TreeSet<SimpleResult> docs;

    public GenericImageSearcher(int maxHits, Class<?> descriptorClass, String fieldName) {
        this.maxHits = maxHits;
        docs = new TreeSet<SimpleResult>();
        this.descriptorClass = descriptorClass;
        this.fieldName = fieldName;
    }

    public ImageSearchHits search(BufferedImage image, IndexReader reader) throws IOException {
        logger.finer("Starting extraction.");
        LireFeature lireFeature = null;
        SimpleImageSearchHits searchHits = null;
        try {
            lireFeature = (LireFeature) descriptorClass.newInstance();
            // Scaling image is especially with the correlogram features very important!
            BufferedImage bimg = image;
            if (Math.max(image.getHeight(), image.getWidth()) > GenericDocumentBuilder.MAX_IMAGE_DIMENSION) {
                bimg = ImageUtils.scaleImage(image, GenericDocumentBuilder.MAX_IMAGE_DIMENSION);
            }
            lireFeature.extract(bimg);
            logger.fine("Extraction from image finished");

            float maxDistance = findSimilar(reader, lireFeature);
            searchHits = new SimpleImageSearchHits(this.docs, maxDistance);
        } catch (InstantiationException e) {
            logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
        } catch (IllegalAccessException e) {
            logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
        }
        return searchHits;
    }

    /**
     * @param reader
     * @param lireFeature
     * @return the maximum distance found for normalizing.
     * @throws java.io.IOException
     */
    protected float findSimilar(IndexReader reader, LireFeature lireFeature) throws IOException {
        float maxDistance = -1f, overallMaxDistance = -1f;
        boolean hasDeletions = reader.hasDeletions();

        // clear result set ...
        docs.clear();

        int docs = reader.numDocs();
        for (int i = 0; i < docs; i++) {
            // bugfix by Roman Kern
            if (hasDeletions && reader.isDeleted(i)) {
                continue;
            }

            Document d = reader.document(i);
            float distance = getDistance(d, lireFeature);
            assert (distance >= 0);
            // calculate the overall max distance to normalize score afterwards
            if (overallMaxDistance < distance) {
                overallMaxDistance = distance;
            }
            // if it is the first document:
            if (maxDistance < 0) {
                maxDistance = distance;
            }
            // if the array is not full yet:
            if (this.docs.size() < maxHits) {
                this.docs.add(new SimpleResult(distance, d));
```

```
                if (distance > maxDistance) maxDistance = distance;
            } else if (distance < maxDistance) {
                // if it is nearer to the sample than at least on of the current set:
                // remove the last one ...
                this.docs.remove(this.docs.last());
                // add the new one ...
                this.docs.add(new SimpleResult(distance, d));
                // and set our new distance border ...
                maxDistance = this.docs.last().getDistance();
            }
        }
        return maxDistance;
    }

    protected float getDistance(Document d, LireFeature lireFeature) {
        float distance = 0f;
        LireFeature lf;
        try {
            lf = (LireFeature) descriptorClass.newInstance();
            String[] cls = d.getValues(fieldName);
            if (cls != null && cls.length > 0) {
                lf.setStringRepresentation(cls[0]);
                distance = lireFeature.getDistance(lf);
            } else {
                logger.warning("No feature stored in this document!");
            }
        } catch (InstantiationException e) {
            logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
        } catch (IllegalAccessException e) {
            logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
        }

        return distance;
    }

    public ImageSearchHits search(Document doc, IndexReader reader) throws IOException {
        SimpleImageSearchHits searchHits = null;
        try {
            LireFeature lireFeature = (LireFeature) descriptorClass.newInstance();

            String[] cls = doc.getValues(fieldName);
            if (cls != null && cls.length > 0)
                lireFeature.setStringRepresentation(cls[0]);
            float maxDistance = findSimilar(reader, lireFeature);

            searchHits = new SimpleImageSearchHits(this.docs, maxDistance);
        } catch (InstantiationException e) {
            logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
        } catch (IllegalAccessException e) {
            logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
        }
        return searchHits;
    }

    public ImageDuplicates findDuplicates(IndexReader reader) throws IOException {
        // get the first document:
        SimpleImageDuplicates simpleImageDuplicates = null;
        try {
            if (!IndexReader.indexExists(reader.directory()))
                throw new FileNotFoundException("No index found at this specific location.");
            Document doc = reader.document(0);

            LireFeature lireFeature = (LireFeature) descriptorClass.newInstance();
            String[] cls = doc.getValues(fieldName);
            if (cls != null && cls.length > 0)
                lireFeature.setStringRepresentation(cls[0]);

            HashMap<Float, List<String>> duplicates = new HashMap<Float, List<String>>();

            // find duplicates ...
            boolean hasDeletions = reader.hasDeletions();

            int docs = reader.numDocs();
            int numDuplicates = 0;
            for (int i = 0; i < docs; i++) {
                if (hasDeletions && reader.isDeleted(i)) {
                    continue;
                }
                Document d = reader.document(i);
                float distance = getDistance(d, lireFeature);

                if (!duplicates.containsKey(distance)) {
                    duplicates.put(distance, new LinkedList<String>());
                } else {
                    numDuplicates++;
                }
                duplicates.get(distance).add(d.getFieldable(DocumentBuilder.FIELD_NAME_IDENTIFIER).stringValue());
            }

            if (numDuplicates == 0) return null;

            LinkedList<List<String>> results = new LinkedList<List<String>>();
```

```
            LinkedList<List<String>> results = new LinkedList<List<String>>();
            for (float f : duplicates.keySet()) {
                if (duplicates.get(f).size() > 1) {
                    results.add(duplicates.get(f));
                }
            }
            simpleImageDuplicates = new SimpleImageDuplicates(results);
        } catch (InstantiationException e) {
            logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
        } catch (IllegalAccessException e) {
            logger.log(Level.SEVERE, "Error instantiating class for generic image searcher: " + e.getMessage());
        }
        return simpleImageDuplicates;

    }

    public String toString() {
        return "GenericSearcher using " + descriptorClass.getName();
    }

}
```

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：

```java
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：

```java
AutoColorCorrelogram acc = new AutoColorCorrelogram(CorrelogramDocumentBuilder.MAXIMUM_DISTANCE)
```

相比而言，更具有通用型。

在search()函数中，调用了一个函数findSimilar()。这个函数的作用是查找相似图片的，分析了一下它的步骤：

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，又调用了一个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：

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

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

```java
public static double getSimilarity(int[] YCoeff1, int[] CbCoeff1, int[] CrCoeff1, int[] YCoeff2, int[] CbCoeff2, int[] CrCoeff2) {
        int numYCoeff1, numYCoeff2, CCoeff1, CCoeff2, YCoeff, CCoeff;

        //Numbers of the Coefficients of two descriptor values.
        numYCoeff1 = YCoeff1.length;
        numYCoeff2 = YCoeff2.length;
        CCoeff1 = CbCoeff1.length;
        CCoeff2 = CbCoeff2.length;

        //take the minimal Coeff-number
        YCoeff = Math.min(numYCoeff1, numYCoeff2);
        CCoeff = Math.min(CCoeff1, CCoeff2);

        setWeightingValues();

        int j;
        int[] sum = new int[3];
        int diff;
        sum[0] = 0;

        for (j = 0; j < YCoeff; j++) {
            diff = (YCoeff1[j] - YCoeff2[j]);
            sum[0] += (weightMatrix[0][j] * diff * diff);
        }

        sum[1] = 0;
        for (j = 0; j < CCoeff; j++) {
            diff = (CbCoeff1[j] - CbCoeff2[j]);
            sum[1] += (weightMatrix[1][j] * diff * diff);
        }

        sum[2] = 0;
        for (j = 0; j < CCoeff; j++) {
            diff = (CrCoeff1[j] - CrCoeff2[j]);
            sum[2] += (weightMatrix[2][j] * diff * diff);
        }

        //returns the distance between the two desciptor values

        return Math.sqrt(sum[0] * 1.0) + Math.sqrt(sum[1] * 1.0) + Math.sqrt(sum[2] * 1.0);
    }
```

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。

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