Zxing库深入学习

前言

> 源码版本：3.3.2
>
> 源码来源：https://github.com/zxing/zxing/tree/master/android

本次调研的二维码扫描的库为Zxing，在进行二维码解析库的选择时对比Zxing和Zbar，发现Zbar的解析效率较Zxing高，但是长期没有人对Zbar的库进行维护，且很难实现本次调研的主要目的（根据图像识别的结果进行自动变焦），所以选择使用Zxing库。

带着问题学习

本次调研需要解决的问题包括如下内容：

1. 详细了解二维码解析的详细流程
2. 尽可能加快二维码解析的速度
3. 实现自动变焦的功能

在这些功能中，我们的主要目的是实现摄像头自动变焦，调研发现，Zxing在解析二维码图片时，会返回解析到特征点信息，且这些信息中带有这些点在我们传入帧图中的相对位置，我们可以根据这些特征点信息计算出二维码在帧图中的大小，然后来控制摄像头进行变焦。

针对解读源码前的相关调研，接下来会带着如下几个问题去阅读Zxing的源码时：

1. 图片流传入Zxing以前需要经过哪些处理？
2. Zxing是如何解析图片的？
3. 特征点是如何获得的？
4. 对于不存在二维码的帧图或者二维码存在不全的帧图，是否能让特征点信息返回？
5. 影响解析效率的因素有哪些？怎么合理优化？

库介绍

本次使用到的库有两个：BGAQRCode-Android、Zxing

根据实际需求，针对库中的代码进行了修改，其中主要包括：

- BGAQRCode-Android

    1. 去掉了与条形码相关的代码
    2. 修改了对焦频率、降低了帧图监听的延迟
    3. 一些与Zxing相关的性能优化（选择）
    4. 添加了部分日志

- Zxing

    1. 添加FoundPartException类型。
    2. 修改FinderPatternFinder的selectBestPatterns()，在发现特征点不全的情况下，抛出异常并返回已经发现的特征点。
    3. 修改Detector类的相关方法，添加 throw FoundPartException。

问题解读

在解读Zxing相关的源码前，我们需要对这几个概念有一定的了解

- 一维码和二维码（来自联图）

一维码就是我们见到的条形码，这种条形码的特点就是它所包含的信息在水平方向上，竖直方向上不包含信息，且一维码包含的信息只能包含字母和数字，且遭到损坏后便不能阅读了。

二维码就是我们这次调研的对象，它的信息存储在水平和竖直两个方向上，比条形码容纳的信息量大，且保密性和可靠性比条形码强，由于二维码中包含有冗余信息，所以就算局部损坏或穿孔，依然可以正确的识别。

- 二值化（来自百度百科）

一幅图像包括目标物体、背景还有噪声，要想从多值的数字图像中直接提取出目标物体，最常用的方法就是设定一个阈值T，用T将图像的数据分成两部分：大于T的像素群和小于T的像素群。这是研究灰度变换的最特殊的方法，称为图像的二值化（BINARIZATION）。

图像的二值化，就是将图像上的像素点的灰度值设置为0或255，也就是将整个图像呈现出明显的只有黑和白的视觉效果。

- 数码变焦和光学变焦（来自ZOL）

简单的理解就是数码变焦是通过放大单位像素来实现放大缩小，光学变焦则是通过镜头的相对位置调整物理焦距实现放大缩小。所以数码变焦是损耗图片质量的，而我们的手机大部分都是数码变焦。

因此，我们实现变焦后传入的图片在质量上有损耗，但是单位像素所包含的信息量也变少了，有助于识别的算法更快、更准确的识别二维码内容。（讲道理，如果放大后的图片能识别到二维码，那不放的也应该是可以的，但是较放大后的图片来说识别度较低，识别速度较慢，会严重影响用户体验）

Zxing解析之前的准备

首先，我们通过预览回调onPreviewFrame方法获取每一帧的图片，然后开线程对图片进行处理，由于异步任务的特性，在task.cancel后并没有真正意义上的停止线程，而onPreviewFrame()的回调是特别频繁的，为了避免多个线程同时执行图像解析方法造成内存问题，在异步线程中加上了同步锁判断，主要方法如下：

ProcessDataTask.class

```
    @Override
    protected String doInBackground(Void... params) {
        if(!BarcodeLockUtil.requestLock()){
            return null;
        }
        Camera.Parameters parameters = mCamera.getParameters();
        Camera.Size size = parameters.getPreviewSize();
        int width = size.width;
        int height = size.height;

//        byte[] rotatedData = new byte[mData.length];
//        for (int y = 0; y < height; y++) {
//            for (int x = 0; x < width; x++) {
//                rotatedData[x * height + height - y - 1] = mData[x + y * width];
//            }
//        }
//        int tmp = width;
//        width = height;
//        height = tmp;

        try {
            if (mDelegate == null) {
                return null;
            }
            return mDelegate.processData(mData,orientation,width, height, false);
        } catch (Exception e1) {
            try {
                return mDelegate.processData(mData,orientation,width, height, true);
            } catch (Exception e2) {
                return null;
            }
        } finally {
            BarcodeLockUtil.releaseLock();
        }
    }
```

接下来就是将图片流传入Zxing的解析接口的工作了，在使用Zxing的接口之前，我们需要对于其需要的几个类有一个认识：

- LuminanceSource
  这个类的子类封装了一些方法来用来获取帧图中的信息，主要方法有:getRow(),getMatrix(),crop()。所得到的都是原图的byte数组，没有对其中的像素点信息进行处理。

- Binarizer
  这个类的子类主要用于将LuminanceSource进行二值化操作并转化成BitArray/BitMatrix类型，主要方法包括：getBlackRow(),getBlackMatrix(),createBinarizer()。

- BinaryBitmap
  这个类是可以看做是对Binarizer的一次再封装，这个类就是一个是解析时直接使用的实体类，其中的所有方法都是公开方法。

- Reader
  这个接口是所有解析类都需要继承的接口，核心方法为两个decode()方法，主要负责对BinaryBitmap中的BitArray进行解析并返回解析结果。

下面是在线程中实际执行解析的方法：

ZxingView.class

```
private QRCodeReader reader = new QRCodeReader();

@Override
public String processData(byte[] data,int orientation,int width, int height, boolean isRetry) {
    Result rawResult = null;
    try {
        PlanarYUVLuminanceSource source = null;
        Rect rect = null;
        if(orientation == BGAQRCodeUtil.ORIENTATION_PORTRAIT) {
            rect = mScanBoxView.getScanBoxAreaRect(width);
            source = new PlanarYUVLuminanceSource(data, width, height, rect.top, rect.left, rect.height(), rect.width(), false);
        }else {
            rect = mScanBoxView.getScanBoxAreaRect(height);
            source = new PlanarYUVLuminanceSource(data,width,height,rect.left,rect.right,rect.width(),rect.height(),false);
        }
        rawResult = reader.decode(new BinaryBitmap(new GlobalHistogramBinarizer(source)));
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        reader.reset();
    }
    if(rawResult != null){
        return rawResult.getText();
    }else {
        return null;
    }
}
```

Zxing解析二维码的步骤&自动变焦的契机

我们将二值化的图片传入Zxing的Detector后，下面两段代码是在解析时最关键的两个类：

Detector.class

```
  public final DetectorResult detect(Map<DecodeHintType,?> hints) throws NotFoundException, FormatException, FoundPartException {

    resultPointCallback = hints == null ? null :
        (ResultPointCallback) hints.get(DecodeHintType.NEED_RESULT_POINT_CALLBACK);

    FinderPatternFinder finder = new FinderPatternFinder(image, resultPointCallback);
    FinderPatternInfo info = finder.find(hints);

    return processFinderPatternInfo(info);
  }
```

FinderPatternFinder.class

```
  final FinderPatternInfo find(Map<DecodeHintType,?> hints) throws NotFoundException, FoundPartException {
    ...

    FinderPattern[] patternInfo = selectBestPatterns();
    ResultPoint.orderBestPatterns(patternInfo);
```

```
    return new FinderPatternInfo(patternInfo);
  }

  private FinderPattern[] selectBestPatterns() throws NotFoundException, FoundPartException{

    int startSize = possibleCenters.size();

    if (startSize < 3) {
      if(startSize > 0){
        FoundPartException exception = FoundPartException.getFoundPartInstance();
        exception.clear();
        for (FinderPattern fp:possibleCenters){
          exception.addPattern(fp);
        }
        throw exception;
      }
      // Couldn't find enough finder patterns
      throw NotFoundException.getNotFoundInstance();
    }

    ...
  }
```

从detect方法中我们可以看到，QRCode的解码过程大致分为两步：找到特征点，然后处理结果（processFinderPatternInfo后面的步骤包括了数据校验和生成最终的矩阵）。调试后发现，find方法只会在识别到了二维码之后才会返回，这明显不是我们需要的，所以，为了实现自动变焦，我们重点需要去改变的是find中的内容。

在find中，开始一大段内容省略的是获取特征点的算法，很复杂，还没有达到能修改这部分算法的层次。但是！它拿到特征点后进行了两个操作：selectBestPatterns()这个方法内部实现，可以发现原来它原来是这么一段：

```
    if (startSize < 3) {
      // Couldn't find enough finder patterns
      throw NotFoundException.getNotFoundInstance();
    }
```

So，当解析到的二维码信息不全时，会直接抛出NotFoundException的异常，我们需要实现自动变焦，就可以考虑在这里拿到它已经找的特征点信息并返回给上层，所以便有了我们上面的代码，在其中插入一段判断，抛出一个不同的异常。其中FoundPartException就是我们自己添加的内容，写法参照了NotFoundException：

```
public class FoundPartException extends ReaderException {

    private static final FoundPartException INSTANCE = new FoundPartException();

    static {
        INSTANCE.setStackTrace(NO_TRACE); // since it's meaningless
    }

    private List<ResultPoint> foundPoints;

    private FoundPartException() {
        // do nothing
        foundPoints = new ArrayList<>();
    }

    public void clear(){
        foundPoints.clear();
    }

    public void addPattern(ResultPoint point) {
        foundPoints.add(point);
    }

    public void addPatterns(List<ResultPoint> points){
        foundPoints.addAll(points);
    }

    public List<ResultPoint> getFoundPoints() {
        return foundPoints == null ? new ArrayList<ResultPoint>() : foundPoints;
    }

    public static FoundPartException getFoundPartInstance() {
        return INSTANCE;
    }
}
```

影响解析效率的因素&能力范围内的优化

在进行解析的过程中，测试发现有如下内容对效率的影响比较明显：

1. 二值化的工具选择
2. 解码格式的选择
3. 图片流的旋转处理
4. 并发线程的影响
5. 解析二维码的算法

其中，我们能够做到的优化是针对1-4的，算法的修改过于麻烦，风险也较大，没有做任何修改。（革命尚未成功，同志仍需努力！）

下面是优化性能/提高代码可读性的内容：

1. 删除了ProcessDataTask中关于图片旋转的方法（这一步很关键，测试发现这一步在大型应用中严重影响效率，下面对异步线程的优化中有详细介绍）
2. 删除了BGAQRCode-Android中与二维码无关的代码
3. 为ProcessDataTask添加了线程锁
4. 解析器使用QRCodeReader（只能识别二维码）
5. 使用GlobalHistogramBinarizer对原数据进行二值化（消耗cpu更低）
6. 提高了自动对焦的频率（有助于变焦后快速识别）

对异步线程的优化

除了加入同步锁，与BGAQRCode-Android相比，这里还做了一个处理，就是去除了竖屏状态下翻转屏幕的算法代码，原因是这段算法在导入到比较大的应用（MOA）中后，执行时间在500ms左右，严重影响到了整体解析速度。

```
byte[] rotatedData = new byte[mData.length];
for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
        rotatedData[x * height + height - y - 1] = mData[x + y * width];
    }
}
int tmp = width;
width = height;
height = tmp;
```

PS：在独立App下面多次测试发现，加入此段代码，在竖屏且无二维码图时，cpu的波动增加了3-5%（通过AS监控波动峰值得到），内存的波动增加了2M（通过AS监控波动峰值得到），最终决定弃用这段代码，通过其他手段去解决横竖问题。

总结

本次针对二维码扫描功能的优化到此就算结束了，本次针对二维码的研究，除了完成任务外，对于二维码、图像识别、Camera相关的内容都有了较深入的学习和了解，总结了这次优化能为我们带来的良性影响如下：

1. 从现有效果来说，与项目中的原有的zbar的解析相比，新的库表现出来的解析速度丝毫不逊色，并且新库在旋转角度、图像变形和远距的场景中表现远远优胜于原来的库。

2. 从长远效果来说，ZXing的库在持续更新，无论是算法的优化还是新的解析格式的添加都是紧跟潮流，想法Zbar的维护工作已经停止，从未来的拓展性来考虑，Zxing的实用性远高于Zbar。

鸣谢

https://github.com/bingoogolapple/BGAQRCode-Android
https://github.com/zxing/zxing
http://www.liantu.com/zhishi/2012070322.html
https://baike.baidu.com/item/%E4%BA%8C%E5%80%BC%E5%8C%96/4766301?fr=aladdin
http://mobile.zol.com.cn/413/4134487_all.html
https://www.cnblogs.com/riasky/p/3508874.html