

相机预览识别 Android SDK 说明文档

V1.0.0.20170510

目录

1.	相机预览识别核心机制.....	2
1.1	相机预览识别核心机制描述.....	2
1.2	相机预览识别核心机制流程图.....	3
2.	识别坐标方法 ISMatch 原理	4
3.	调整预览框位置和大小.....	5
4.	相机预览识别对焦原理.....	6
5.	相机预览识别选择最佳分辨率.....	6
6.	历史修订记录.....	7

1. 相机预览识别核心机制

1.1 相机预览识别核心机制描述

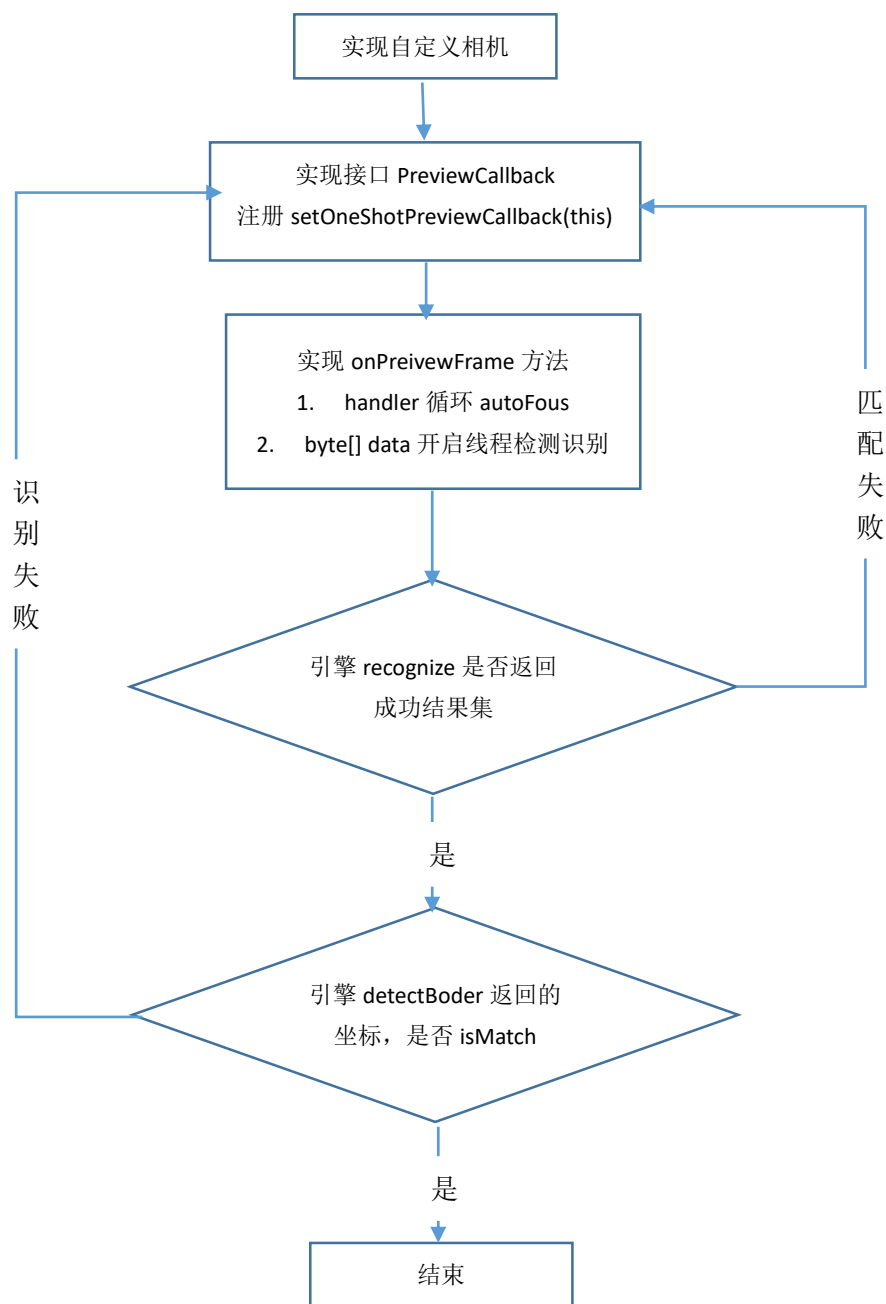
PreviewActivity 是自定义相机的核心类,核心机制分以下几点:

- 第一步: 实现关键的 Camera.PreviewCallback 接口, 实现 onPreviewFrame(byte[] data, Camera camera)方法, 这个方法会将每一帧的结果回调。
- 第二步: 在 onResume 中添加 onPreviewFrame 的回调注册, mCamera.setOneShotPreviewCallback(this)。
- 第三步: 在 onPreviewFrame 中用 handler.post 配合 mCamera.autoFous(focusCallBack)来循环对焦。
- 第四步: 在 onPreviewFrame 中开启一个全局线程 DetectThread, 并且将 data 帧数据添加到线程队列里面, 然后对数据循环过滤匹配再进行识别, 具体流程:
 - ✧ DetectThread 里面拿到 data 帧数据之后,调用引擎 detectBorder 方法, 返回当前预览图片对应的屏幕四个点的坐标, 然后我们调用 isMatch 判断当前预览图的坐标是否在可识别坐标范围内 (isMatch 方法机制会在下面章节【[识别坐标方法 ISMatch 原理](#)】具体介绍)。
 - ✧ 如果 isMatch 验证通过, 即当前预览坐标匹配识别的位置, 然后才会调

用引擎 recognize 进行识别，识别成功返回结果则结束扫描退出当前 Activity，识别失败则继续预览扫描，需要继续调用 `mCamera.setOneShotPreviewCallback(this)`。

✧ 如果 `isMatch` 验证不通过，则继续预览扫描，需要继续调用 `mCamera.setOneShotPreviewCallback(this)`。

1.2 相机预览识别核心机制流程图



2. 识别坐标方法 ISMatch 原理

- isMatch 方法的声明定义：`public boolean isMatch(int left, int top, int right, int bottom, int[] qua)`
- ✧ 输入参数：left top right bottom 是预览框的坐标，这个可以通过下面【[调整预览框位置和大小](#)】详细讲解如何通过 `getPositionWithArea` 方法获取。
- ✧ 输入参数：int[] qua 是预览框的坐标，另外 int[] qua 是引擎 `int[] out = mIDCardScanSDK.detectBorder(data, width,height, (int) top, (int) (height - right), (int) bottom, (int) (height - left))` 返回的坐标，`detectBorder` 的输入参数 data 是预览的帧数据，width,height 是当前预览界面的宽高，其他的 left top right bottom 是预览框的坐标获取和上面一样。
- isMatch 方法的核心原理
 - ✧ 首先定义一个阈值，用来判断可识别范围的值，经过长期测试我们推荐使用 `float dif=120`，客户可以根据自己的需求调试更改该值。
 - ✧ 预览的每帧数据的四个坐标点的坐标分别和预览框的四个坐标点进行判断，如果差值的绝对值在 dif 范围之内，我们就累计一个点达标
 - ✧ 当 4 个点累计两个以上，我们就认为当前预览的图片在可识别范围之内，则该方法返回 true 否则返回 false
 - ✧ 详细代码参考 `PreviewActivity` 中的 `isMatch` 方法

3. 调整预览框位置和大小

➤ `getPositionWithArea` 方法的声明定义：`public Map<String, Float>getPositionWithArea(int newWidth, int newHeight, float scaleW, float scaleH);`

✧ 输入参数：`newWidth newHeight` 分别是当前预览框的分辨率的宽高，注意横竖屏宽高值切换，然后 `scaleW scaleH` 是用户在手机屏幕上 draw 预览框 UI 时使用，`scaleW = getWidth() / (float) previewHeight;`
`scaleH = getHeight() / (float) previewWidth;` 具体怎么 draw 这个 ui 详情见 `PreviewActivity` 中的 `DetectView` 类，当 `isMatch` 和 `detectBorder` 调用的时候 `scaleW scaleH` 都等于 1。

➤ 如何调整预览框位置和大小，该部分代码比较关键，见下面代码：

```
public Map<String, Float> getPositionWithArea(int newWidth, int
newHeight, float scaleW, float scaleH) {
    float left, top, right, bottom;
    Map<String, Float> map = new HashMap<String, Float>();
    if (isVertical) { // vertical
        float dis = 1 / 16f;
        left = newWidth * dis;
        right = newWidth - left;
        top = 200f * scaleH;
        bottom = top + (newWidth - left - left) * 0.618f;
        /**
         * 注解部分是：竖直模式居中，如果是竖直模式相对位置 则需要注意 不同
         分辨率手机的缩放比例
         */
        // top = (newHeight - (newWidth - left - left) * 0.618f) / 2;
        // bottom = newHeight - top;
    }
}
```

- ✧ 首先调整预览框 ui, 传入的 `scaleW` 和 `scaleH` 必须是通过屏幕宽高和预览分辨率的比例, 否则会出现实际传入引擎解析的预览区域和实际预览框的区域不一致, 最终导致识别效果差。
- ✧ 调整预览框的大小, 主要调整 `dis` 这个参数
- ✧ 预览框的宽高比例是应该按照实体证件高宽比例 0.618, 所以建议客户严格执行, 切勿随意修改
- ✧ 调整预览框的位置 则是更改 `left right top bottom` 相关的值, 可以参考代码中的修改方式

4. 相机预览识别对焦原理

- `PreviewActivity` 中使用的是通过 `onPreviewFrame` 里启动 `mHandler.sendMessageDelayed(MSG_AUTO_FOCUS, 100)`; 这个方法只会第一次调用。
- 在 `handler` 的回调里面调用相机对焦 `mCamera.autofocus(callback)` 然后在 `callback` 里面记录当前是对焦成功或者失败, 然后 `mHandler.sendMessageDelayed(MSG_AUTO_FOCUS, 1000)`, 这样就会每隔一秒对焦一次。对焦是否成功也可以作为是否进行识别的校验变量, 这样可以保证进入识别的图片的清晰度。

5. 相机预览识别选择最佳分辨率

- `PreviewActivity` 中有个方法是获取最佳分辨率, 然后作为相机的预览分辨率可以避免预览的效果图片扭曲, 与真实物体比例不匹配, `public Size getOptimalPreviewSize(List<Size> sizes, int w, int h, int targetHeight)`。
- `getOptimalPreviewSize` 原理就是将相机支持的所有 `size` 列表, 和当前使

用机型的宽高，来筛选比例最合适的

- targetHeight 是我们自定义的标准宽，基于长期的识别效果及稳定性，我们推荐 720 是一个识别较稳定的分辨率。
- 核心代码请参考 PreviewActivity 的 getOptimalPreviewSize 方法。

6. 历史修订记录

更新日期	更新内容	更新版本
2017.05.10	首次创建	V.1.0.0.20170510