**ZBar鸿蒙开源第三方组件——Zbar\_ohos条形码阅读器**

## 前言

基于安卓平台的条形码阅读器控件ZBar（https://github.com/ZBar/ZBar），实现了鸿蒙化迁移和重构，代码已经开源到（https://gitee.com/isrc\_ohos/ZBar），欢迎各位下载使用并提出宝贵意见！

## 背景

Zbar-ohos是基于鸿蒙系统的条形码阅读器，支持EAN-13 / UPC-A、UPC-E、EAN-8、Code 128、CODE39、Codabar和QR码的识别，目前已经广泛应用于扫码登记、扫码观影、扫码登录等多个领域。

## 组件效果展示

1. 添加权限

打开软件后，会显示如图1所示的添加摄像头权限提示。点击“始终允许”按钮，并重启该软件（刷新UI界面），即可扫描条形码。



图1 扫描二维码界面

1. 扫描效果

扫描界面包含两个部分：对准器和状态栏。对准器显示摄像头拍摄的画面，条形码需要置于此范围内，才可以被扫描。状态栏用于显示当前的扫描状态或扫描结果。

（1）一维条形码扫描

一维条形码一般是在水平方向上表达信息，而在垂直方向不表达任何信息。为了方便对准器的读取，其高度通常是固定的。

ZBar组件扫描一维条形码的效果图2所示。摄像头扫到条形码时，下方状态栏的显示内容由“扫描中”更新为条形码的扫描结果。扫到下一个条码时，状态栏的扫描结果也实时更新。



图2 条形码扫描结果

（2）二维条形码扫描

二维条形码在水平和垂直方向上都表示信息，信息容量大，结构通常为方形结构，保密级别高，可直接显示英文、中文、数字、符号、图型等。

ZBar组件扫描二维条形码的效果图3所示。扫描过程与上述一维条形码一致，状态栏会显示二维条形码的扫描结果。



图3 二维码扫描结果

## ****Sample解析****

Sample部分首先创建相机设备并合理配置，然后将相机获得的原始数据传递给Library扫描处理，最后获取扫描结果并显示在屏幕上。下面对Sample部分的代码进行具体解释：

1. 生成Camera类对象

CameraKit类可以提供使用相机功能的条目，CameraStateCallbackImpl 类是相机创建和相机运行时的回调。此处通过CameraKit类来生成Camera对象，不同寻常的是，CameraKit类并没有将Camera对象直接返回，而是需要从CameraStateCallbackImpl 回调中获取。

**private void** openCamera(){  
 *// 获取 CameraKit 对象* **cameraKit** = CameraKit.*getInstance*(**this**);  
 **if** (**cameraKit** == **null**) {  
 **return**;  
 }  
 **try** {  
 *// 获取当前设备的逻辑相机列表cameraIds* String[] cameraIds = **cameraKit**.getCameraIds();  
 **if** (cameraIds.**length** <= 0) {  
 System.***out***.println(**"cameraIds size is 0"**);  
 }  
 *// 用于相机创建和相机运行的回调* CameraStateCallbackImpl cameraStateCallback = **new** CameraStateCallbackImpl();  
 **if**(cameraStateCallback ==**null**) {  
 System.***out***.println(**"cameraStateCallback is null"**);  
 }  
 *// 创建用于运行相机的线程* EventHandler eventHandler = **new** EventHandler(EventRunner.*create*(**"CameraCb"**));  
 **if**(eventHandler ==**null**) {  
 System.***out***.println(**"eventHandler is null"**);  
 }  
 *// 创建相机* **cameraKit**.createCamera(cameraIds[0], cameraStateCallback, eventHandler);  
 } **catch** (IllegalStateException e) {  
 System.***out***.println(**"getCameraIds fail"**);  
 }  
}

2、绑定相机的Surface

Surface用于实现相机的预览、拍照、录像等功能。此处为相机添加：previewSurface和 dataSurface。前者用来展示相机拍摄到的界面；后者用来读取并处理相机拍摄到的数据信息。

**private final class** CameraStateCallbackImpl **extends** CameraStateCallback {  
 *// 相机创建和相机运行时的回调* @Override

**public void** onCreated(Camera camera) {  
 **mcamera** = camera;*//获取到Camera 对象* CameraConfig.Builder cameraConfigBuilder = camera.getCameraConfigBuilder();  
 **if** (cameraConfigBuilder == **null**) {  
 System.***out***.println(**"onCreated cameraConfigBuilder is null"**);  
 **return**;  
 }  
 *// 配置预览的 Surface* cameraConfigBuilder.addSurface(**previewSurface**);  
 *// 配置处理数据的Surface* **dataSurface** = **imageReceiver**.getRecevingSurface();  
 cameraConfigBuilder.addSurface(**dataSurface**);  
 **try** {  
 *// 相机设备配置* camera.configure(cameraConfigBuilder.build());  
 } **catch** (IllegalArgumentException e) {  
 System.***out***.println(**"Argument Exception"**);  
 } **catch** (IllegalStateException e) {  
 System.***out***.println(**"State Exception"**);  
 }  
 }

}

3、开启循环帧捕获

用户一般在画面生成后，才执行拍照或者其他操作。开启循环帧捕获后，dataSurface可以获得来自相机的数据。

@Override  
 **public void** onConfigured(Camera camera) {

*// 获取预览配置模板*   
 FrameConfig.Builder frameConfigBuilder = **mcamera**.getFrameConfigBuilder(***FRAME\_CONFIG\_PREVIEW***);  
 *// 配置预览 Surface* frameConfigBuilder.addSurface(**previewSurface**);  
 *// 配置拍照的 Surface* frameConfigBuilder.addSurface(**dataSurface**);  
**try** {  
 *// 启动循环帧捕获* **int** triggerId = **mcamera**.triggerLoopingCapture(frameConfigBuilder.build());  
 } **catch** (IllegalArgumentException e) {  
 System.***out***.println(**"Argument Exception"**);  
 } **catch** (IllegalStateException e) {  
 System.***out***.println(**"State Exception"**);  
 *}*

*}*

4、扫描相机数据

dataSurface中的数据为相机原始数据，其格式为YUV420，需要将其封装为Image类的数据才能执行传入ImageScanner类进行正式扫描。

*// 相机原始数据封装为Image数据*

Image barcode = **new** Image(mImage.getImageSize().***width***,mImage.getImageSize().***height***, **"Y800"**);

barcode.setData(YUV\_DATA);

*//Image数据扫描*

**int** result = **scanner**.scanImage(barcode);

1. 显示预览数据的扫描结果

由于对准器中的条形码可能不止一个，ImageScanner类的扫描结果可能也有多个，因此最后返回的扫描结果是SymbolSet类型，此数据类型是可以盛纳多个Symbol数据的容器，每个Symbol数据代表一个条形码的扫描结果。

*//创建可以盛纳多个Symbol数据的容器SymbolSet*

SymbolSet syms = **scanner**.getResults();  
**for** (Symbol sym : syms) { *//遍历SymbolSet 中的每个元素* **handler**.postTask(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **scanText**.setText(**"扫描结果:"** + sym.getData());*//获取Symbol中的信息*  
 **scanText**.invalidate();  
 }  
 });

## Library解析

Library部分主要是对dataSurface的数据进行扫描，此处主要涉及两个功能：（1）相机原始数据封装为Image数据；（2）对Image数据进行扫描。由于这部分主要由C语言实现，所以此处只解析大概原理，展示主要接口，不再进行底层代码的展示。

1. 相机原始数据封装为Image数据

Image支持多种数据格式，包括常见的YUV以及RGB数据。此处需要的Image数据是“Y800”类型或者“GRAY”类型，即条形码的扫描数据仅需要图像的灰度数据。

**public native void** setData(**byte**[] data);

1. 对Image数据进行扫描

使用scanImage()方法对传入的Image数据进行扫描。该过程首先对传入的图像进行配置校验，然后以一个像素点为增量逐行扫描，扫描路径为Z字型，并且完成对扫描数据的滤波，求取边缘梯度，梯度阈值自适应，确定边缘等操作，最后将扫描数据转化成明暗宽度流。 通过明暗宽度流的变化格律可以知道当前正在被扫描的条形码的种类，然后依据固定的解码方法进行解码，便可得到条形码信息。

**public native int** scanImage(Image image);

## 项目贡献人

陈丛笑 郑森文 朱伟 陈美汝 张馨心