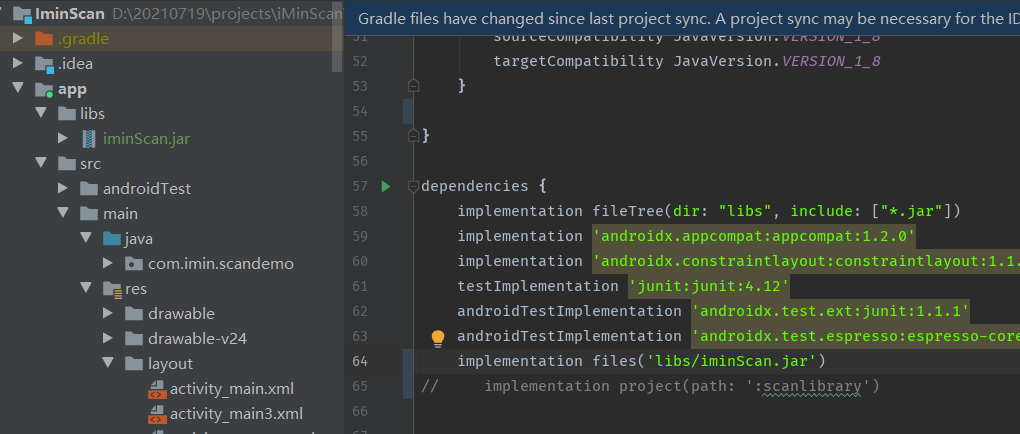
# IMIN QR code scanning

###### How to use IMIN's QR code scanning sdk

1. Initialize the response configuration by having the developer's scanning application inherit the SDK's scanning interface, CaptureActivity. Initialize the relevant configurations and retrieve the return values.
2. Create your own camera interface and use the SDK to perform image parsing. This approach is relatively more complex but provides greater flexibility.

**First, import the IMIN QR code scanning SDK into your project as shown below:**

##### The first usage method :

To simplify development, the "imin" QR code SDK has encapsulated a scanning module. Developers need to inherit CaptureActivity in the calling interface, configure the scanning settings, and receive the return values in the onScanResultCallback(Result result) method.

@SuppressLint("NewApi")

public class CustomCaptureActivity extends CaptureActivity {

StringBuilder sb = new StringBuilder();

private Toast toast;

private TextView textresult;

public int decode\_count = 0;

@Override

public int getLayoutId() {

return R.layout.custom\_capture\_activity;

}

@Override

public void onCreate(Bundle icicle) {

super.onCreate(icicle);

// Toolbar toolbar = findViewById(R.id.toolbar);

textresult = findViewById(R.id.textresult);

}

@Override

public void initCameraScan() {//Initializing the camera.

super.initCameraScan();

//Initializing the decoding configuration.

DecodeConfig decodeConfig = new DecodeConfig();

decodeConfig.setHints(DecodeFormatManager.ALL\_HINTS)////Setting up decoding.

.setSupportVerticalCode(true)//Setting whether to support scanning vertical barcodes. （Improving recognition accuracy will also increase performance consumption accordingly.）

.setSupportLuminanceInvert(true)//Setting whether to support the recognition of inverted colors, where black and white colors are reversed.（Improving recognition accuracy will also increase performance consumption accordingly.）

.setAreaRectRatio(0.8f)//Setting the recognition area ratio, defaulting to 0.8. The set ratio will ultimately define a rectangle based on this proportion within the preview area for scanning recognition.

// .setAreaRectVerticalOffset(0)// Setting the recognition area vertical offset, which defaults to 0. A value of 0 means centered, and it can be a negative number as well.

// .setAreaRectHorizontalOffset(0)// Setting the recognition area horizontal offset, which defaults to 0. A value of 0 means centered, and it can be a negative number as well.

.setFullAreaScan(false);//Setting whether to enable full-area recognition, with the default value being false.

// Get CameraScan, there are QR code related configuration settings inside. CameraScan contains some methods that support chain calls, which means that the calls return CameraScan itself. It is recommended to configure some of these settings before calling startCamera.

getCameraScan().setPlayBeep(true)//Set whether to play sound effects, with the default value being false.

.setPlayRaw(R.raw.beep)

.setVibrate(true)// Set whether vibration is enabled, with the default value being false.

// .setCameraConfig(new CameraConfig())//Set camera configuration information, where CameraConfig can override the options method for custom configuration.

.setCameraConfig(new ResolutionCameraConfig(CustomCaptureActivity.this))// Configure CameraConfig to customize settings according to your own needs.

.setNeedAutoZoom(false)//Automatically scale down small QR codes; the default value is false.

.setNeedTouchZoom(false)// Support multi-touch pinch-to-zoom; the default value is true.

.setDarkLightLux(45f)// Set the threshold for low light conditions (unit: lux). This requires binding a flashlight using {@link #bindFlashlightView(View)} for it to take effect.

.setBrightLightLux(100f)// Set the threshold for sufficiently bright light conditions (unit: lux). This requires binding a flashlight using {@link #bindFlashlightView(View)} to take effect.

.bindFlashlightView(null)// Bind flashlight: Once bound, the flashlight button can be dynamically shown or hidden based on the light sensor.

.setOnScanResultCallback(this)//Set the callback for scan results. You can set a callback to handle scan results on your own or when continuous scanning is needed, and manage the related logic accordingly.

.setAnalyzer(new MultiFormatAnalyzer(decodeConfig))//Set the analyzer. DecodeConfig can be used to configure decoding settings. If the built-in options do not meet your needs, you can also implement custom ones.

.setAnalyzeImage(true);//Set whether to analyze images, with the default value being true. If set to false, it's equivalent to disabling the QR code recognition functionality.

}

/\*\*

\* QR code scanning result callback.

\* @param result

\* @return Returning false indicates no interception. This will close the scanning interface and return the result to the calling interface.；

\* Returning true indicates interception, and you need to handle the logic yourself. When isAnalyze is set to true, it will continue analyzing images by default (i.e., continuous scanning).

\* If you only want to intercept the QR code scanning result callback and do not want to continue analyzing images (disabling continuous scanning), you can achieve this by implementing the interception logic at the point where you handle the scanning results.

\* Use{@link CameraScan#setAnalyzeImage(boolean)}，

\* Because the {@link CameraScan#setAnalyzeImage(boolean)} method allows you to dynamically control whether to continue analyzing images. \*

\*/

@Override

public boolean onScanResultCallback(Result result) {

Resources resources = getResources();

sb.append(resources.getString(R.string.count) + decode\_count++);

sb.append("\n" + resources.getString(R.string.time\_consuming) + (System.currentTimeMillis()-result.getTimestamp()) + " ms\n");

sb.append(resources.getString(R.string.symbology) +result.getBarcodeFormat().name()+ "\n");

sb.append(resources.getString(R.string.capacity) + result.getNumBits()+ "\n");

sb.append(resources.getString(R.string.content) + result.toString());

textresult.setText(sb.toString());

sb.delete(0, sb.length());

/\*

\* Because the setAnalyzeImage method allows dynamic control over whether to continue analyzing images.

\*

\* 1. Because analyzing images is set to true by default, if you want to support continuous scanning, you can simply return true.

\* When the processing logic for continuous scanning is complex, please call getCameraScan().setAnalyzeImage(false) before handling the logic.

\* To stop analyzing images, after completing the logic processing, call getCameraScan().setAnalyzeImage(true) to resume analyzing images.

\*

\* 2. If you only want to intercept the QR code scanning result callback and handle the logic yourself, but you don't want to continue analyzing images (i.e., you don't want continuous scanning), you can achieve this by using:

\* Call getCameraScan().setAnalyzeImage(false) to stop analyzing images.

\*/

return true;

}

@Override

public void onScanResultFailure() {// Scan failure callback.

textresult.setText(sb.toString());

sb.delete(0, sb.length());

}

private void showToast(String text){

if(toast == null){

toast = Toast.makeText(this,text,Toast.LENGTH\_SHORT);

}else{

toast.setDuration(Toast.LENGTH\_SHORT);

toast.setText(text);

}

toast.show();

}

}

##### The second usage method:

After integrating the iminScan.jar,

1.Import the relevant Java interface classes into your business logic code.

import com.google.zxing.Result;

import com.imin.scan.ScanUtils;

import com.imin.scan.Symbol;

2.Initialization of relevant parameters:

private ScanUtils scanUtils;

// Obtaining the decoding utility class.

scanUtils = ScanUtils.getInstance(this);

scanUtils.initScan();//Initialize

scanUtils.initBeepSound(true,R.raw.beep);//Initialize the audio files. You can customize the audio file for scan recognition.

scanUtils.setConfig(Symbol.ALL\_FORMATS);// Allow all decoding types

// Decode types in detail

public interface Symbol {

int ALL\_FORMATS = 0;// All are supported

int ONE\_D\_FORMATS = 20; // One-dimensional barcode

int QR\_CODE\_FORMATS = 21;// QR code

int DATA\_MATRIX\_FORMATS = 22;//DATA\_MATRIX

int PRODUCT\_FORMATS = 23;//PRODUCT

/\*\* Aztec 2D barcode format. \*/

int AZTEC = 1;

/\*\* CODABAR 1D format. \*/

int CODABAR = 2;

/\*\* Code 39 1D format. \*/

int CODE\_39 = 3;

/\*\* Code 93 1D format. \*/

int CODE\_93 = 4;

/\*\* Code 128 1D format. \*/

int CODE\_128 = 5;

/\*\* Data Matrix 2D barcode format. \*/

int DATA\_MATRIX = 6;

/\*\* EAN-8 1D format. \*/

int EAN\_8 = 7;

/\*\* EAN-13 1D format. \*/

int EAN\_13 = 8;

/\*\* ITF (Interleaved Two of Five) 1D format. \*/

int ITF = 9;

/\*\* MaxiCode 2D barcode format. \*/

int MAXICODE = 10;

/\*\* PDF417 format. \*/

int PDF\_417 = 11;

/\*\* QR Code 2D barcode format. \*/

int QR\_CODE = 12;

/\*\* RSS 14 \*/

int RSS\_14 = 13;

/\*\* RSS EXPANDED \*/

int RSS\_EXPANDED = 14;

/\*\* UPC-A 1D format. \*/

int UPC\_A = 15;

/\*\* UPC-E 1D format. \*/

int UPC\_E = 16;

/\*\* UPC/EAN extension format. Not a stand-alone format. \*/

int UPC\_EAN\_EXTENSION = 17;

}

3.Image decoding, if using the Android system camera for decoding, can be done in the image preview callback function.

Call the decoding of preview data directly in the PreviewCallback.onPreviewFrame(byte[] data, Camera camera) method.

Or in the ImageReader.onImageAvailable(ImageReader reader) method...

Use

Image image = reader.acquireLatestImage();

//We can convert this frame data into a byte array, similar to the preview frame data in the Camera1's PreviewCallback callback.

ByteBuffer buffer = image.getPlanes()[0].getBuffer();

byte[] data = new byte[buffer.remaining()];

The method involves converting the obtained data into raw data for decoding.

// The filled data 'data' corresponds to the raw camera data, while 'previewSize\_width' and 'previewSize\_height' represent the current set width and height of the preview image.

Result rawResult = scanUtils.getScanResult(data,previewSize\_width,previewSize\_height);//

if (scanUtils.getNsyms() != 0 && rawResult != null) {// Decoding

}

4. Retrieve the decoding result and barcode type.

if (scanUtils.getNsyms() != 0 && rawResult != null) {

Log.i("imin", " Barcode type:"+rawResult.getBarcodeFormat().name());//Barcode type, such as...“EAN-8”

Log.i("imin","result:"+rawResult.toString())//Decoded result string

}

More instructions

The QR code types currently supported by the iMin QR code SDK include the following:

* One-dimensional barcodes: EAN-8, EAN-13, UPC-A, UPC-E, CODABAR, Code39, Code93, Code128, RSS\_14, ITF, RSS\_EXPANDED，UPC\_EAN\_EXTENSION
* Two-dimensional barcodes: QR Code , PDF417，DataMatrix，AZTEC，MAXICODE