安卓中使用相机从来就不是一件容易的事。

Camera1要自己管理Camera相机实例，要处理SufraceView相关的一堆东西，还有预览尺寸跟画面尺寸的选择，页面生命周期切换等等问题。。。

后来推出了Camera2，从官方Demo
就上千行代码来看，Camera2并不解决用起来复杂的问题，它提供了更多的调用接口，可定制性更好，结果就是对普通开发者来说更难用了。。。

终于Google也意识到这个问题，推出了最终版CameraX. CameraX实际上还是用的Camera2，但它对调用API进行了很好的封装，使用起来非常方便。官方教程也很详细，如下：
https://codelabs.developers.google.com/codelabs/camerax-getting-started/#0

官方用的Kotlin代码，我转成了Java，其实用起来差不多。

> 注意：CameraX跟Camera2一样最低支持API21，也就是5.0及以上。
> 开发环境用Android Studio3.3及以上，依赖库都用androidx的

1 导入依赖

在app的build.gradle中加入

```
    implementation 'androidx.appcompat:appcompat:1.1.0-rc01'

    // Use the most recent version of CameraX, currently that is alpha04
    def camerax_version = "1.0.0-alpha04"
    implementation "androidx.camera:camera-core:${camerax_version}"
    implementation "androidx.camera:camera-camera2:${camerax_version}"
```

2 布局文件和权限

放一个TextureView就行了

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity">

    <TextureView
            android:id="@+id/view_finder"
            android:layout_width="640px"
            android:layout_height="640px"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintEnd_toEndOf="parent" />
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

权在AndroidManifest.xml中加入相机权限

```
<uses-permission android:name="android.permission.CAMERA" />
```

并加入动态申请权限代码，这里省略掉（你要在App安装后手动打开相机权限）。

3 启动相机

给TextureView设置布局变化的监听，用updateTransform()更新相机预览，然后startCamera()启动相机

```
        TextureView viewFinder = findViewById(R.id.view_finder);
        viewFinder.addOnLayoutChangeListener(new View.OnLayoutChangeListener() {
            @Override
            public void onLayoutChange(View view, int i, int i1, int i2, int i3,
int i4, int i5, int i6, int i7) {
                updateTransform();
            }
        });

        viewFinder.post(new Runnable() {
            @Override
            public void run() {
                startCamera();
            }
        });
```

更新相机预览：主要是给TextureView设置一个旋转的矩阵变化，防止预览方向不对

```
    private void updateTransform() {
        Matrix matrix = new Matrix();
        // Compute the center of the view finder
        float centerX = viewFinder.getWidth() / 2f;
        float centerY = viewFinder.getHeight() / 2f;

        float[] rotations = {0,90,180,270};
        // Correct preview output to account for display rotation
        float rotationDegrees =
rotations[viewFinder.getDisplay().getRotation()];

        matrix.postRotate(-rotationDegrees, centerX, centerY);

        // Finally, apply transformations to our TextureView
        viewFinder.setTransform(matrix);
    }
```

启动相机：创建PreviewConfig和Preview这两个对象，可以设置预览图像的尺寸和比例，在OnPreviewOutputUpdateListener回调中用setSurfaceTexture方法，将相机图像输出到TextureView。最后用CameraX.bindToLifecycle方法将相机与当前页面的生命周期绑定。

```
    private void startCamera() {
        // 1. preview
        PreviewConfig previewConfig = new PreviewConfig.Builder()
                .setTargetAspectRatio(new Rational(1, 1))
                .setTargetResolution(new Size(640,640))
                .build();

        Preview preview = new Preview(previewConfig);
        preview.setOnPreviewOutputUpdateListener(new
Preview.OnPreviewOutputUpdateListener() {
            @Override
            public void onUpdated(Preview.PreviewOutput output) {
                ViewGroup parent = (ViewGroup) viewFinder.getParent();
                parent.removeView(viewFinder);
                parent.addView(viewFinder, 0);

                viewFinder.setSurfaceTexture(output.getSurfaceTexture());
                updateTransform();
            }
        });

        CameraX.bindToLifecycle(this, preview);
```

这样就实现了基本的相机预览功能。这几个方法都很简单明了，对外只依赖一个TextureView。生命周期自动绑定，这意味着代码可以写在一块，在一处调用。不像以前这里插一段代码，那里插一段代码。

还有最大的好处，就是可扩展性。相机预览使用了PreviewConfig和Preview两个对象，加入新的相机功能同样是加两个对象XXXConfig和XXX，其他地方都不同改！

加入拍照功能就加入ImageCaptureConfig和ImageCapture，加入图像分析功能就加入ImageAnalysisConfig和ImageAnalysis，非常方便统一。

4 拍照

创建ImageCaptureConfig和ImageCapture这两个对象，用imageCapture.takePicture方法传入相片保存地址就行了。当然在生命周期绑定中也加上imageCapture。

> ImageCaptureConfig可以定制相片尺寸和长宽比例，这里的尺寸和比例跟相机预览的尺寸比例无关，我测试传入任何比例都能得到图片。

```
        // 2. capture
        ImageCaptureConfig imageCaptureConfig = new ImageCaptureConfig.Builder()
                .setTargetAspectRatio(new Rational(1,1))
                .setCaptureMode(ImageCapture.CaptureMode.MIN_LATENCY)
                .build();
        final ImageCapture imageCapture = new ImageCapture(imageCaptureConfig);
        viewFinder.setOnLongClickListener(new View.OnLongClickListener() {
            @Override
            public boolean onLongClick(View view) {
```

```
                File photo = new File(getExternalCacheDir() + "/" +
System.currentTimeMillis() + ".jpg");
                imageCapture.takePicture(photo, new
ImageCapture.OnImageSavedListener() {
                    @Override
                    public void onImageSaved(@NonNull File file) {
                        showToast("saved " + file.getAbsolutePath());
                    }

                    @Override
                    public void onError(@NonNull ImageCapture.UseCaseError
useCaseError, @NonNull String message, @Nullable Throwable cause) {
                        showToast("error " + message);
                        cause.printStackTrace();
                    }
                });
                return true;
            }
        });

        CameraX.bindToLifecycle(this, preview, imageCapture);
```

5 图片分析

图片分析名字很高大上，实际上就是图像数据回调，实时获取相机的图像数据，可以自己处理这些图像。

创建ImageAnalysisConfig和ImageAnalysis这两个对象，创建一个HandlerThread用于在子线程中处理数据，创建一个ImageAnalysis.Analyzer接口实现类，在analyze(ImageProxy imageProxy, int rotationDegrees)回调方法中就能拿到图像数据了。当然ImageAnalysis对象也要绑定生命周期。

> 我这里分析图像数据用了之前写的一个工具YUVDetectView，来分析图像属于哪种YUV420格式。

```
        // 3. analyze
        HandlerThread handlerThread = new HandlerThread("Analyze-thread");
        handlerThread.start();

        ImageAnalysisConfig imageAnalysisConfig = new
ImageAnalysisConfig.Builder()
                .setCallbackHandler(new Handler(handlerThread.getLooper()))

.setImageReaderMode(ImageAnalysis.ImageReaderMode.ACQUIRE_LATEST_IMAGE)
                .setTargetAspectRatio(new Rational(2, 3))
//                .setTargetResolution(new Size(600, 600))
                .build();

        ImageAnalysis imageAnalysis = new ImageAnalysis(imageAnalysisConfig);
        imageAnalysis.setAnalyzer(new MyAnalyzer());

        CameraX.bindToLifecycle(this, preview, imageCapture, imageAnalysis);



    private class MyAnalyzer implements ImageAnalysis.Analyzer {
```

```
        @Override
        public void analyze(ImageProxy imageProxy, int rotationDegrees) {
            final Image image = imageProxy.getImage();
            if(image != null) {
                Log.d("chao", image.getWidth() + "," + image.getHeight());
                imageView.input(image);
            }
        }
    }
```

Github地址

https://github.com/rome753/android-YuvTools