最新 (/)
团队号 (/channel/index?type=263&name=team)
移动开发 (/channel/index?type=272&name=mobile)
Java (/channel/index?type=257&name=java)
Python (/channel/index?type=268&name=python)
个人号 (/channel/index?type=271&name=person)
前端 (/channel/index?type=264&name=web)
后端 (/channel/index?type=269&name=server)
测试 (/channel/index?type=266&name=test)
更多 (/channel/index?type=270&name=other)

QQ空间终端开发团队 (/author/index?authorId=222)

11月17日 1新/0 阅读： 961 ： 0

简介：

Android相机开发那些坑

2016-01-28 22:59 zakiwang

最近我负责开发了一个跟Android相机有关的需求，新功能允许用户使用手机摄像头，快速拍摄特定尺寸（1：1或3：4）的照片，并支持在拍摄出的照片上做贴纸相关的操作。由于之前没有接触过Android相机开发，所以在整个开发过程中踩了不少坑，费了不少时间和精力。这篇文章总结了Android相机开发的相关知识、流程，以及容易遇到的坑，希望能帮助今后可能会接触Android相机开发的朋友快速上手，节省时间，少走弯路。

一.Android中开发相机应用的两种方式

Android系统提供了两种使用手机相机资源实现拍摄功能的方法，一种是直接通过Intent调用系统相机组件，这种方法快速方便，适用于直接获得照片的场景，如上传相册，微博、朋友圈发照片等。另一种是使用相机API来定制自定义相机，这种方法适用于需要定制相机界面或者开发特殊相机功能的场景，如需要对照片做裁剪、滤镜处理，添加贴纸，表情，地点标签等。这篇文章主要是从如何使用相机API来定制自定义相机这个方向展开的。

二.相机API中关键类解析

通过相机API实现拍摄功能涉及以下几个关键类和接口：

Camera：最主要的类，用于管理和操作camera资源。它提供了完整的相机底层接口，支持相机资源切换，设置预览/拍摄尺寸，设定光圈、曝光、聚焦等相关参数，获取预览/拍摄帧数据等功能，主要方法有以下这些：

- open()：获取camera实例。
- setPreviewDisplay(SurfaceHolder)：绑定绘制预览图像的surface。surface是指向屏幕窗口原始图像缓冲区（raw buffer）的一个句柄，通过它可以获得这块屏幕上对应的canvas，进而完成在屏幕上绘制View的工作。通过surfaceHolder可以将Camera和surface连接起来，当camera和surface连接后，camera获得的预览帧数据就可以通过surface显示在屏幕上了

- setPrameters设置相机参数，包括前后摄像头，闪光灯模式、聚焦模式、预览和拍照尺寸等。
- startPreview():开始预览，将camera底层硬件传来的预览帧数据显示在绑定的surface上。
- stopPreview():停止预览，关闭camra底层的帧数据传递以及surface上的绘制。
- release():释放Camera实例
- takePicture(Camera.ShutterCallback shutter, Camera.PictureCallback raw, Camera.PictureCallback jpeg):这个是实现相机拍照的主要方法，包含了三个回调参数。shutter是快门按下时的回调，raw是获取拍照原始数据的回调，jpeg是获取经过压缩成jpg格式的图像数据的回调。

SurfaceView：用于绘制相机预览图像的类，提供给用户实时的预览图像。普通的view以及派生类都是共享同一个surface的，所有的绘制都必须在UI线程中进行。而surfaceview是一种比较特殊的view，它并不与其他普通view共享surface，而是在内部持有了一个独立的surface,surfaceview负责管理这个surface的格式、尺寸以及显示位置。由于UI线程还要同时处理其他交互逻辑，因此对view的更新速度和帧率无法保证，而surfaceview由于持有一个独立的surface，因而可以在独立的线程中进行绘制，因此可以提供更高的帧率。自定义相机的预览图像由于对更新速度和帧率要求比较高，所以比较适合用surfaceview来显示。

SurfaceHolder：surfaceholder是控制surface的一个抽象接口，它能够控制surface的尺寸和格式，修改surface的像素，监视surface的变化等等，surfaceholder的典型应用就是用于surfaceview中。surfaceview通过getHolder()方法获得surfaceholder 实例，通过后者管理监听surface 的状态。

SurfaceHolder.Callback接口：负责监听surface状态变化的接口，有三个方法：

- surfaceCreated(SurfaceHolder holder)：在surface创建后立即被调用。在开发自定义相机时，可以通过重载这个函数调用camera.open()、camera.setPreviewDisplay()，来实现获取相机资源、连接camera和surface等操作。
- surfaceChanged(SurfaceHolder holder, int format, int width, int height):在surface发生format或size变化时调用。在开发自定义相机时，可以通过重载这个函数调用camera.startPreview来开启相机预览，使得camera预览帧数据可以传递给surface，从而实时显示相机预览图像。

最新 (/)

团队号 (/channel/index?type=263&name=team)

移动开发 (/channel/index?type=272&name=mobile)

Java (/channel/index?type=257&name=java)

Python (/channel/index?type=268&name=python)

个人号 (/channel/index?type=271&name=person)

前端 (/channel/index?type=264&name=web)

后端 (/channel/index?type=269&name=server)

测试 (/channel/index?type=266&name=test)

更多 (/channel/index?type=270&name=other)

- surfaceDestroyed(SurfaceHolder holder)：在surface销毁之前被调用。在开发自定义相机时，可以通过重载这个函数调用camera.stopPreview()，camera.release()来实现停止相机预览及释放相机资源等操作。

三.自定义相机的开发过程

定制一个自定义相机应用，通常需要完成以下步骤，其流程图如图1所示：

- 检测并访问相机资源 检查手机是否存在相机资源，如果存在，请求访问相机资源。
- 创建预览类 创建继承自SurfaceView并实现SurfaceHolder接口的拍摄预览类。此类能够显示相机的实时预览图像。
- 建立预览布局 有了拍摄预览类，即可创建一个布局文件，将预览画面与设计好的用户界面控件融合在一起。
- 设置拍照监听器 给用户界面控件绑定监听器，使其能响应用户操作（如按下按钮），开始拍照过程。
- 拍照并保存文件 将拍摄获得的图像转换成位图文件，最终输出保存成各种常用格式的图片。
- 释放相机资源 相机是一个共享资源，必须对其生命周期进行细心的管理。当相机使用完毕后，应用程序必须正确地将其释放，以免其它程序访问使用时，发生冲突。

最新 (/)

团队号 (/channel/index?type=263&name=team)

移动开发 (/channel/index?type=272&name=mobile)

Java (/channel/index?type=257&name=java)

Python (/channel/index?type=268&name=python)

个人号 (/channel/index?type=271&name=person)

前端 (/channel/index?type=264&name=web)

后端 (/channel/index?type=269&name=server)

测试 (/channel/index?type=266&name=test)

更多 (/channel/index?type=270&name=other)

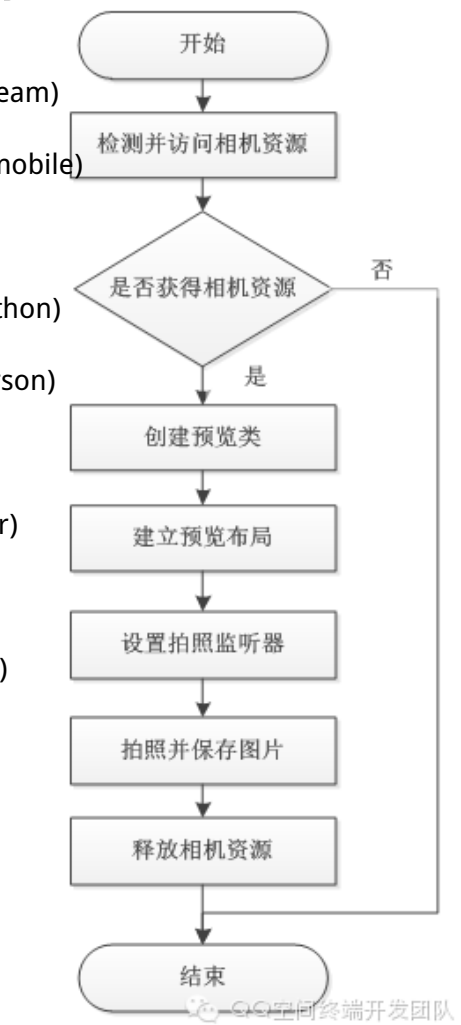


图1 定制自定义相机的过程

对应到代码编写上可以分成三个步骤：

第一步：在AndroidManifest.xml中添加Camera相关功能使用的权限，具体声明有以下这些：

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-feature android:name="android.hardware.camera"/>
<uses-feature android:name="android.hardware.ca
<uses-permission android:name="android.permission.
```


第二步：编写相机操作功能类CameraOperationHelper。采用单例模式来统一管理相机资源，封装相机API的直接调用，并提供用于跟自定义相机Activity做UI交互的回调接口，其功能函数如下，主要有创建\释放相机，连接\开始\关闭预览界面，拍照，自动对焦，切换前后摄像头，切换闪光灯模式等，具体实现可以参考官方API文档。

最新 (/)

团队号 (/channel/index?type=263&name=team)

移动开发 (/channel/index?type=272&name=mobile)

Java (/channel/index?type=257&name=java)

Python (/channel/index?type=268&name=python)

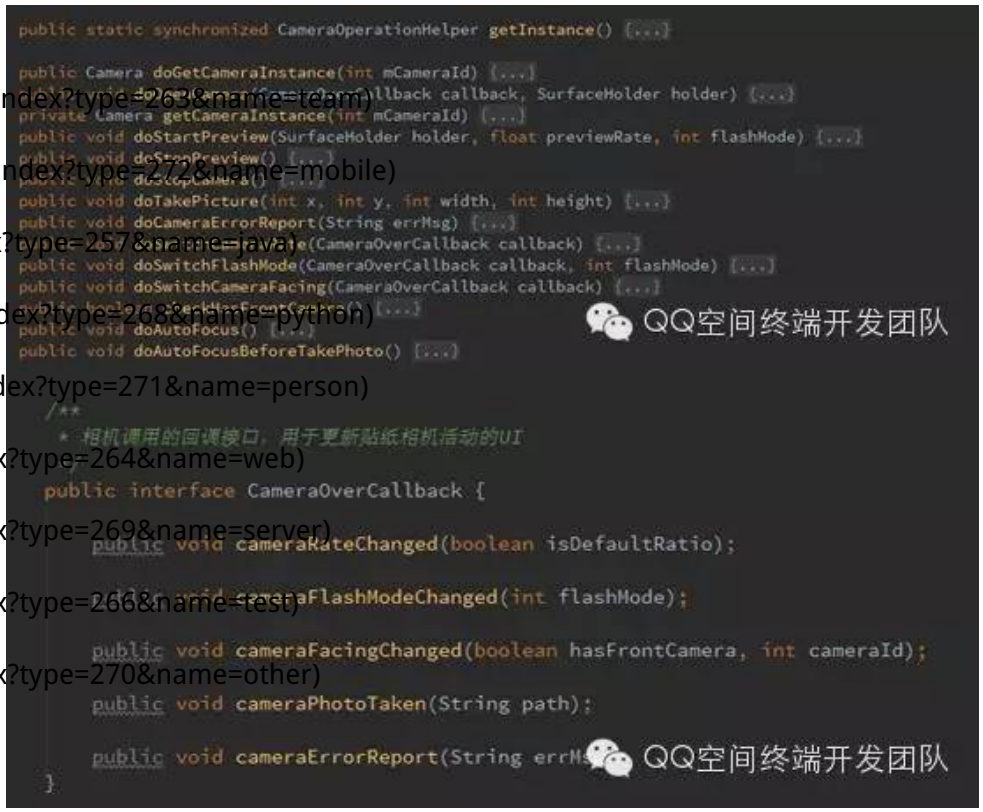
个人号 (/channel/index?type=271&name=person)

前端 (/channel/index?type=264&name=web)

后端 (/channel/index?type=269&name=server)

测试 (/channel/index?type=266&name=test)

更多 (/channel/index?type=270&name=other)



第三步：编写自定义相机Activity，主要是定制相机界面，实现UI交互逻辑，如按钮点击事件处理，icon资源切换，镜头尺寸切换动画等。这里需要声明一个SurfaceView对象来实时显示相机预览画面。通过SurfaceHolder及其Callback接口来一同管理屏幕surface和相机资源的连接，相机预览图像的显示/关闭。

四. 开发过程遇到的一些坑

下面再讲讲我在开发自定义相机时踩过的一些坑：

1. Activity设为竖屏时，SurfaceView预览图像颠倒90度。

说明这个问题之前，先介绍下Android手机上几个方向的概念：

屏幕方向：在Android系统中，屏幕的左上角是坐标系统的原点（0,0）坐标。原点向右延伸是X轴正方向，原点向下延伸是Y轴正方向。

相机传感器方向：手机相机的图像数据都是来自于摄像头硬件的图像传感器，这个传

最新 (/)

团队号 (/channel/index?type=263&name=team)

移动开发 (/channel/index?type=272&name=mobile)

Java (/channel/index?type=257&name=java)

Python (/channel/index?type=268&name=python)

个人号 (/channel/index?type=271&name=person)

前端 (/channel/index?type=264&name=web)

后端 (/channel/index?type=269&name=server)

测试 (/channel/index?type=266&name=test)

更多 (/channel/index?type=270&name=other)

感器在被固定到手机上后有一个默认的取景方向，如下图2所示，坐标原点位于手机横放时的左上角，即与横屏应用的屏幕X方向一致。换句话说，与竖屏应用的屏幕X方向呈90度角。

图2 相机传感器方向示意图

相机的预览方向：由于手机屏幕可以360度旋转，为了保证用户无论怎么旋转手机都能看到“正确”的预览画面（这个“正确”是指显示在UI预览界面的画面与人眼看到的眼前的画面是一致的），Android系统底层根据当前手机屏幕的方向对图像传感器采集到的数据进行了旋转处理，然后才送给显示系统，因此可以保证预览画面始终“正确”。在相机API中可以通过setDisplayOrientation()设置相机预览方向。在默认情况下，这个值为0，与图像传感器一致。因此对于横屏应用来说，由于屏幕方向和预览方向一致，预览图像不会颠倒90度。但是对于竖屏应用，屏幕方向和预览方向垂直，所以会出现颠倒90度现象。为了得到正确的预览画面，必须通过API将相机的预览方向旋转90，保持与屏幕方向一致，如图3所示。

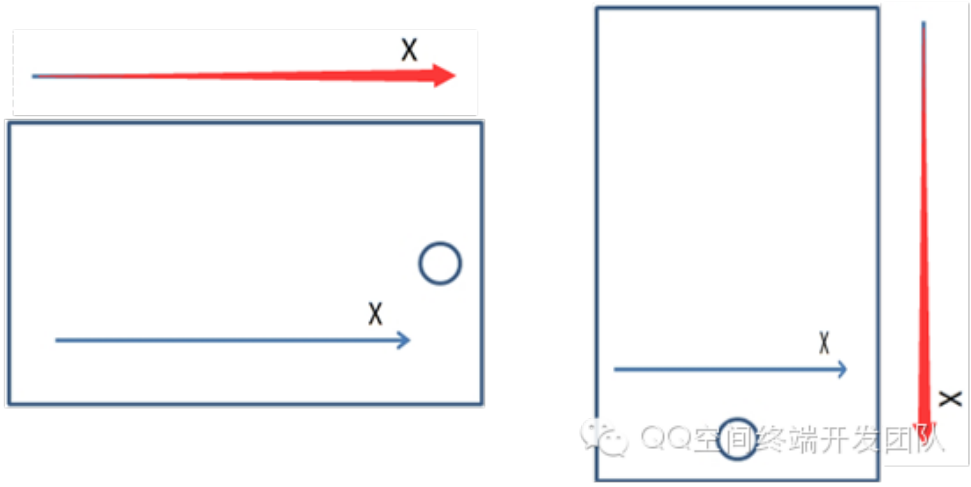


图3 相机预览方向示意图

（红色箭头为预览方向，蓝色方向为屏幕方向）

最新 (/)

团队号 (/channel/index?type=263&name=team)

移动开发 (/channel/index?type=272&name=mobile)

Java (/channel/index?type=257&name=java)

Python (/channel/index?type=268&name=python)

个人号 (/channel/index?type=271&name=person)

前端 (/channel/index?type=264&name=web)

后端 (/channel/index?type=269&name=server)

测试 (/channel/index?type=266&name=test)

更多 (/channel/index?type=270&name=other)

相机的拍照方向：当点击拍照按钮，拍摄的照片是由图像传感器采集到的数据直接存储到SDCard上产生的，因此，相机的拍照方向与传感器方向是一致的。

2. SurfaceView预览图像、拍摄照片拉伸变形

说明这个问题之前，同样先说一下几个跟相机有关的尺寸。

SurfaceView尺寸：即自定义相机应用中用于显示相机预览图像的View的尺寸，当它铺满全屏时就是屏幕的大小。这里surfaceview显示的预览图像暂且称作手机预览图像。

Previewsize：相机硬件提供的预览帧数据尺寸。预览帧数据传递给SurfaceView，实现预览图像的显示。这里预览帧数据对应的预览图像暂且称作相机预览图像。

Picturesize：相机硬件提供的拍摄帧数据尺寸。拍摄帧数据可以生成位图文件，最终保存成.jpg或者.png等格式的图片。这里拍摄帧数据对应的图像称作相机拍摄图像。图4说明了以上几种图像及照片之间的关系。手机预览图像是直接提供给用户看的图像，它由相机预览图像生成，拍摄照片的数据则来自于相机拍摄图像。

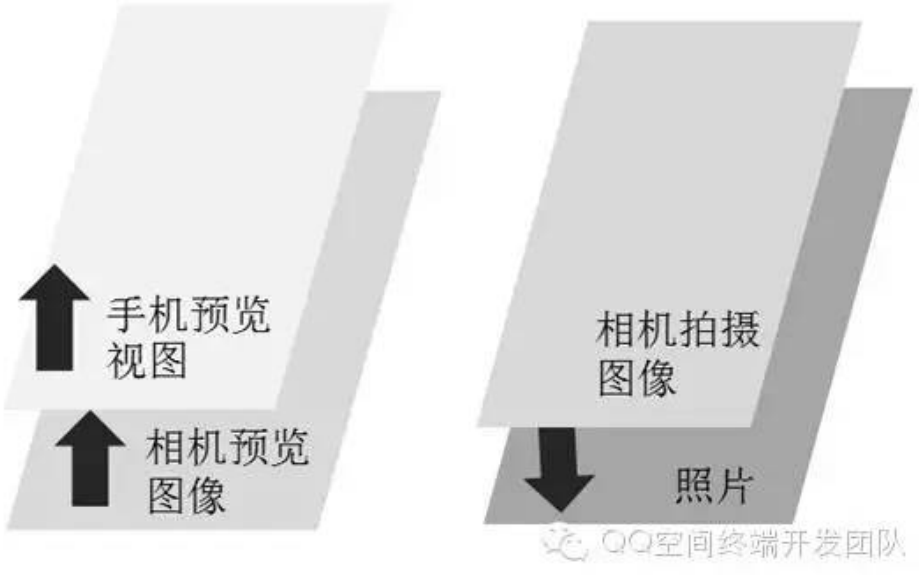


图4 几种图像之间的关系

下面说下我在开发过程中遇到的三种拉伸变形现象：

1、手机预览画面中物体被拉伸变形。

2、拍摄照片中物体被拉伸变形。

3、点击拍照瞬间，手机预览画面会停顿下，此时的图像是拉伸变形的，然后预览画面恢复后图像又正常了。

现象1的原因是SurfaceView的宽高比例与Previewsize的宽高比例不一致。因为手机预览视图的图

像是由相机预览图像根据SurfaceView大小缩放得来的，当长宽比不一致时必然会导致图像变形。后两个现象的原因则是Previewsize和Picturesize的长宽比率不一致所致，查了相关的资料，发现其具体原因跟某些手机相机硬件的底层实现有关。总之为了避免以上几种变形现象的发生，在开发时最好将SurfaceView、PreviewSize、PictureSize三个尺寸保证长宽比例一致。具体实现可以先通过camera.getSupportedPreviewSizes()和camera.getSupportedPictureSizes()获得相机硬件支持的所有预览和拍摄尺寸，然后在里面筛选出和SurfaceView的长宽比一致并且大小合适的尺寸，通过camera.setPrameters来更新设置。注意：市场上手机相机硬件支持的尺寸一般都是主流的4:3或者16:9，所以SurfaceView尺寸不能太奇葩，最好也设置成这样的长宽比。

3. 各种crash

```
java.lang.RuntimeException: autoFocus failed
    at android.hardware.Camera.native_autoFocus(Native Method)
    at android.hardware.Camera.autoFocus(Camera.java:1453)
    at com.qzonex.module.pastercamera.CameraOperatic.Helper.c(ProGuard:191)
    at com.qzonex.module.pastercamera.CameraOperation.Helper.a(ProGuard:334)
    at com.qzonex.module.pastercamera.QzonePasterCameraActivity.onClick(ProGuard:276)
java.lang.RuntimeException: takePicture failed
    at android.hardware.Camera.native_takePicture(Native Method)
    at android.hardware.Camera.takePicture(Camera.java:1617)
    at android.hardware.Camera.takePicture(Cam .java:1558)
    at com.qzonex.module.pastercamera.a.onAutoFocus(ProGuard:58)
java.lang.IllegalArgumentException: x + width must be <= bitmap.width()
    at android.graphics.Bitmap.createBitmap(Bitmap.java:666)
    at com.qzonex.module.pastercamera.ImageUtil.a(ProGuard:35)
    at com.qzonex.module.pastercamera.CameraOpe_ ionHelper.b(ProGuard:276)
    at com.qzonex.module.pastercamera.CameraOperationHelper.b(ProGuard:30)
```

前两个Crash的原因是：相机硬件在聚焦和拍照前必须要保证已经连接到surface，并且开启相机预览，surface有收到预览数据。如果在还没有执行camera. setPreviewDisplay或者未调用camera. startPreview之前,就调用camera.autofocus或camera.takepicture，就会出现这个运行时异常。对应到自定义相机的代码中，要注意在拍照按钮事件响应中执行camera.autofocus或camera.takepicture前，一定要检验camera有没有设置预览Surfaceview并开启了相机预览。这里有个方法可以判断预览状态：Camera.setPreviewCallback是预览帧数据的回调函数，它会在SurfaceView收到相机的预览帧数据时被调用，因此在里面可以设置是否允许对焦和拍照的标志位。

```
mCamera.setPreviewCallback((data, camera) -> {
        mdata = data;
        isPreviewIntialized = true;//被获取到surfaceView
        mSafeToTakePhoto = true;//可以拍照
});
```


还有一点要注意，camera.takePicture()在执行过程中会执行camera.stopPreview来获取拍摄帧数据，表现为预览画面卡住，而如果此时用户点击了按钮的话，也就是调用camera.takepicture，也会出现上面的crash，因此在开发时，可能还需要屏蔽拍照按钮的连续点击。

最新 (/)

团队号 (/channel/index?type=263&name=team)

移动开发 (/channel/index?type=272&name=mobile)

Java (/channel/index?type=257&name=java)

Python (/channel/index?type=268&name=python)

个人号 (/channel/index?type=271&name=person)

前端 (/channel/index?type=264&name=web)

后端 (/channel/index?type=269&name=server)

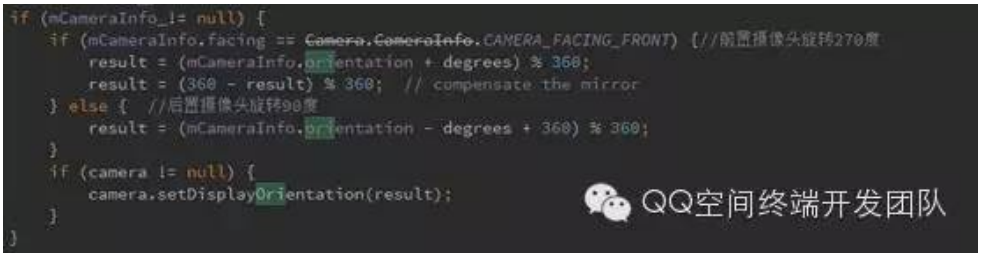
测试 (/channel/index?type=266&name=test)

更多 (/channel/index?type=270&name=other)

第三个crash则涉及图像的裁剪，由于要支持1：1或者4：3尺寸镜头，所以会需要对预览视图进行裁剪。由于是竖屏应用，所以裁剪区域的坐标系跟相机传感器方向是成90度角的，表现在裁剪里就是，屏幕上的x方向，对应在拍摄图像上是高度方向，而屏幕上的y方向，对应到拍摄图像上则是宽度方向。因此在计算时要一定注意坐标系的转换以及越界保护。

4. 前置摄像头的镜像效果

Android相机硬件有个特殊设定，就是对于前置摄像头，在展示预览视图时采用类似镜面的效果，显示的是摄像头成像的镜像。而拍摄出的照片则仍采用摄像头成像。看到这里，大家可能会有些怀疑，不妨现在就试试自己Android手机上的前置摄像头，对比下预览图像和拍摄出照片的区别。这是由于底层相机在传递前置摄像头预览数据时做了水平翻转变换，即将x方向镜像翻转180度。这个变化对之前竖屏预览的方向也会造成影响，本来对于后置摄像头旋转90度即可使预览视图正确，而对前置摄像头，如果也旋转90度的话，看到的预览图像则是上下颠倒的（因为x方向翻转了180度），因此必须再旋转180度，才能显示正确，如图5所示，大家可以结合之前相机预览方向的示意图一起理解。



最新 (/)

团队号 (/channel/index?type=263&name=team)

移动开发 (/channel/index?type=272&name=mobile)

Java (/channel/index?type=257&name=java)

Python (/channel/index?type=268&name=python)

个人号 (/channel/index?type=271&name=person)

前端 (/channel/index?type=264&name=web)

后端 (/channel/index?type=269&name=server)

测试 (/channel/index?type=266&name=test)

更多 (/channel/index?type=270&name=other)

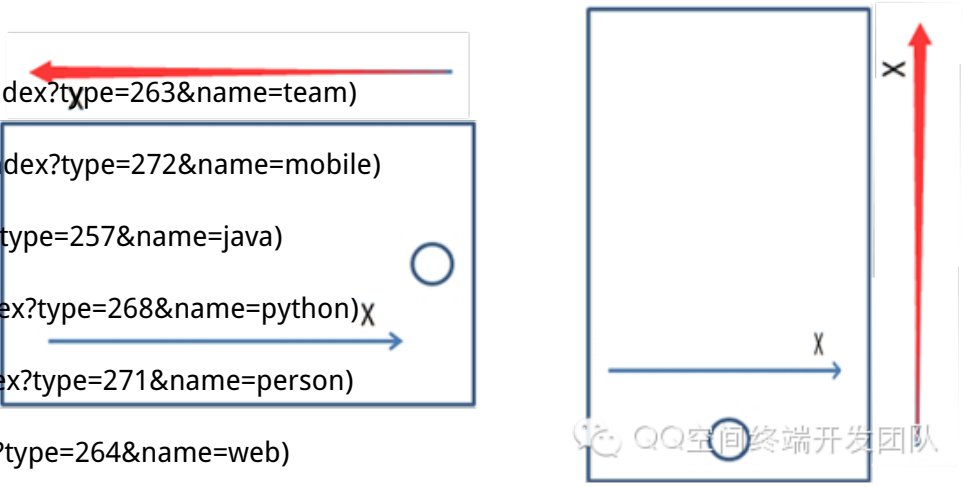


图5 前置摄像头的预览方向示意图

此外，由于拍摄图像并没有做水平翻转，所以对于前置摄像头拍出来的照片，用户会发现跟预览时所见的是左右翻转的。这个在一定程度上会影响用户体验。为了解决这个问题，可以对前置摄像头拍摄的图像在生成位图文件时增加一个水平翻转矩阵变换。

5. 锁屏下相机资源的释放问题

为了节省手机电量，不浪费相机资源，在开发的自定义相机里，如果预览图像已不需要显示，如按Home键盘切换后台或者锁屏后，此时就应该关闭预览并把相机资源释放掉。参考官方API文档，当surfaceView变成可见时，会创建surface并触发surfaceHolder.callback接口中surfaceCreated回调函数。而surfaceview变成不可见时，则会销毁surface，并触发surfacedestroyed回调函数。我们可以在对应的回调函数里，处理相机的相关操作，如连接surface、开启/关闭预览。 至于相机资源释放，则可以放在Acticity的onpause里执行。相应的，要重新恢复预览图像时，可以把相机资源申请和初始化放在Acticity的onResume里执行，然后通过创建surfaceview，将camera和surface相连并开启预览。

但是在开发过程中发现，对于按HOME键切后台场景，程序可以正常运行。对于锁屏场景，则在重新申请相机资源时会发生crash，说相机资源访问失败。那么原因是什么呢？我在代码里增加了调试log， 检查了代码的执行顺序，结果如下：

在自定义相机页面按HOME键时的执行流程：

程序运行->按HOME键

Activity调用的顺序是onPause->onStop

SurfaceView调用了surfaceDestroyed方法

最新 (/)

团队号 (/channel/index?type=263&name=team)

移动开发 (/channel/index?type=272&name=mobile)

Java (/channel/index?type=25?&name=java)

Python (/channel/index?type=268&name=python)

个人号 (/channel/index?type=271&name=person)

前端 (/channel/index?type=264&name=web)

后端 (/channel/index?type=269&name=server)

测试 (/channel/index?type=266&name=test)

更多 (/channel/index?type=270&name=other)

然后再切回程序

Activity调用的顺序是onRestart->onStart->onResume

SurfaceView调用了surfaceCreated->surfaceChanged方法

而对于锁屏，其执行流程则是：

Activity调用onPause方法

解锁后Activity调用onResume方法

SurfaceView中surfaceholder.callback的所有方法都没有执行

问题找到了，由于锁屏时，callback的回调方法没有执行，导致相机和预览的连接还没有断开，相机资源就被释放了，所以导致在重新申请相机资源时，系统报crash。根据上面的文档，推测是锁屏下系统并没有改变surfaceview的可见性，于是我尝试在onPause和onResume时通过手动设置surfaceview的visibile属性，结果发现可以正常触发回调函数了。

由于在切后台或者锁屏时，用户本来就应该看不到surfaceview，因此这种手动更改surfaceview的可见性的方法，并不会对用户的体验造成影响。

```
protected void onResume() {
    super.onResume();
    CameraOperationHelper.getInstance().doGetCameraInstance(mCameraId);
    if (needSetVisible && mSurfaceView != null) {
        mSurfaceView.setVisibility(View.VISIBLE);
    } else {
        needSetVisible = true;
    }
    if (mCaptureBtn != null && !mCaptureBtn.isEnabled()) {
        mCaptureBtn.setEnabled(true);
    }
}

protected void onPause() {
    CameraOperationHelper.getInstance().doStopCamera();//释放相机资源
    CameraOperationHelper.getInstance().release();
    if (needSetVisible && mSurfaceView != null) {
        mSurfaceView.setVisibility(View.INVISIBLE);
    }
    super.onPause();
}
```

最新 (/)

团队号 (/channel/index?type=263&name=team)

移动开发 (/channel/index?type=272&name=mobile)

Java (/channel/index?type=257&name=java)

Python (/channel/index?type=268&name=python)

个人号 (/channel/index?type=271&name=person)

前端 (/channel/index?type=264&name=web)

后端 (/channel/index?type=269&name=server)

测试 (/channel/index?type=266&name=test)

更多 (/channel/index?type=270&name=other)

iOS高性能图片架构与设计 (/html/222/201510/207840007/1.html)

Android GC 那点事 (/html/222/201510/400021278/1.html)

【React Native for Android】框架启动核心路径剖析 (/html/222/201604/2649796767/1.html)

QQ空间直播秒开优化实践 (/html/222/201606/2649796799/1.html)

猜你喜欢

我是不是该安静的睡去？ (/html/300/201602/403080333/1.html)

再见了仙童！安森美半导体24亿美元正式收购Fairchild (/html/282/201511/400363630/1.html)

Mac下提升工作效率的方式 (/html/233/201605/2650264099/1.html)

轻松一下：一些Haskell语言的漫画 (/html/411/201607/2247483892/1.html)

然而大部分工程师的期权并没有什么用 (/html/175/201509/210671786/1.html)