

官网帮助文档链接：  
 http://developer.android.com/guide/components/fragments.html

主要看两张图，和跑代码

一，Fragment的生命周



二，与Activity生命周期的对比



场景演示 : 切换到该Fragment

11-29 14:26:35.095: D/AppListFragment(7649): onAttach  
11-29 14:26:35.095: D/AppListFragment(7649): onCreate  
11-29 14:26:35.095: D/AppListFragment(7649): onCreateView  
11-29 14:26:35.100: D/AppListFragment(7649): onActivityCreated  
11-29 14:26:35.120: D/AppListFragment(7649): onStart  
11-29 14:26:35.120: D/AppListFragment(7649): onResume

屏幕灭掉：

11-29 14:27:35.185: D/AppListFragment(7649): onPause  
11-29 14:27:35.205: D/AppListFragment(7649): onSaveInstanceState  
11-29 14:27:35.205: D/AppListFragment(7649): onStop

屏幕解锁

11-29 14:33:13.240: D/AppListFragment(7649): onStart  
11-29 14:33:13.275: D/AppListFragment(7649): onResume

切换到其他Fragment:  
11-29 14:33:33.655: D/AppListFragment(7649): onPause  
11-29 14:33:33.655: D/AppListFragment(7649): onStop  
11-29 14:33:33.660: D/AppListFragment(7649): onDestroyView

切换回本身的Fragment:

11-29 14:33:55.820: D/AppListFragment(7649): onCreateView  
11-29 14:33:55.825: D/AppListFragment(7649): onActivityCreated  
11-29 14:33:55.825: D/AppListFragment(7649): onStart  
11-29 14:33:55.825: D/AppListFragment(7649): onResume

回到桌面

11-29 14:34:26.590: D/AppListFragment(7649): onPause  
11-29 14:34:26.880: D/AppListFragment(7649): onSaveInstanceState  
11-29 14:34:26.880: D/AppListFragment(7649): onStop

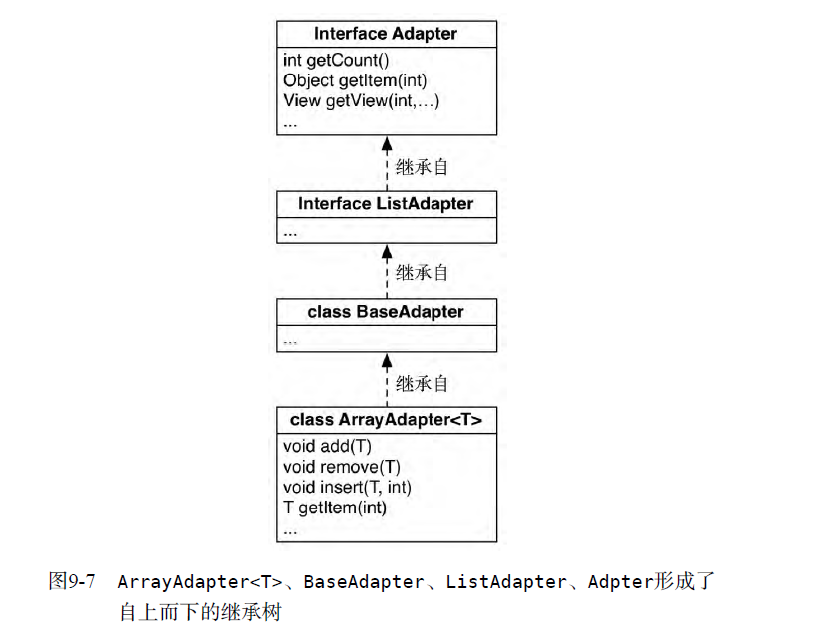
回到应用

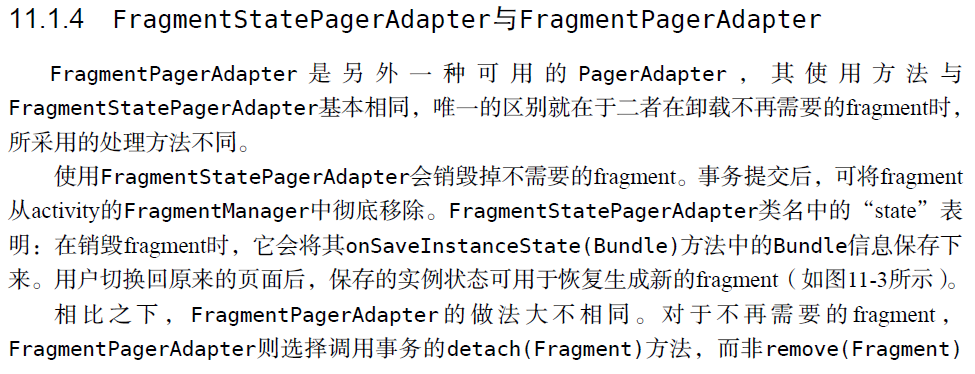
11-29 14:36:51.940: D/AppListFragment(7649): onStart  
11-29 14:36:51.940: D/AppListFragment(7649): onResume

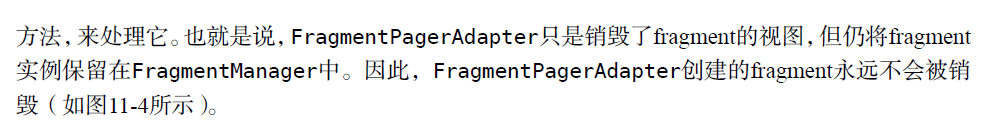
退出应用

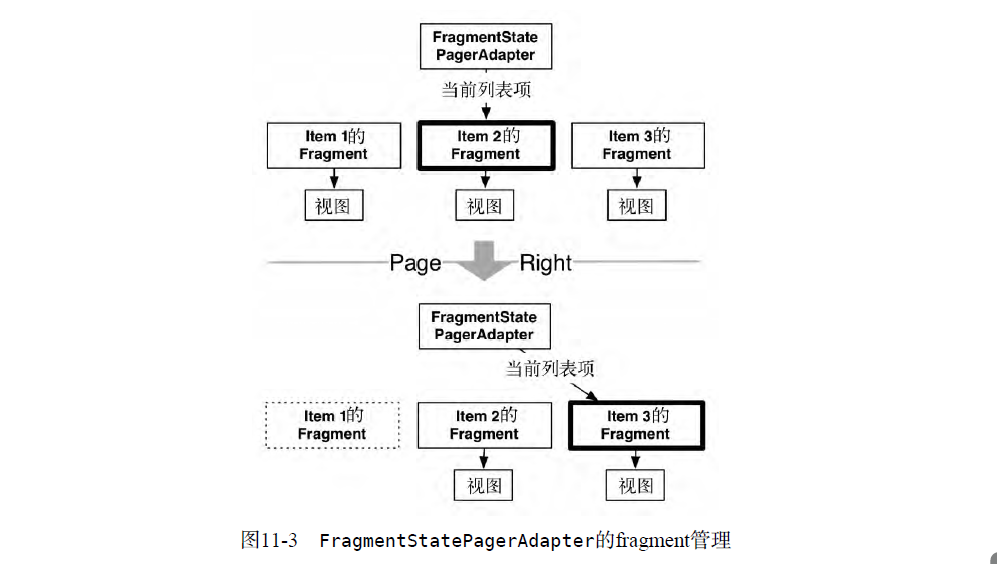
11-29 14:37:03.020: D/AppListFragment(7649): onPause  
11-29 14:37:03.155: D/AppListFragment(7649): onStop  
11-29 14:37:03.155: D/AppListFragment(7649): onDestroyView  
11-29 14:37:03.165: D/AppListFragment(7649): onDestroy  
11-29 14:37:03.165: D/AppListFragment(7649): onDetach

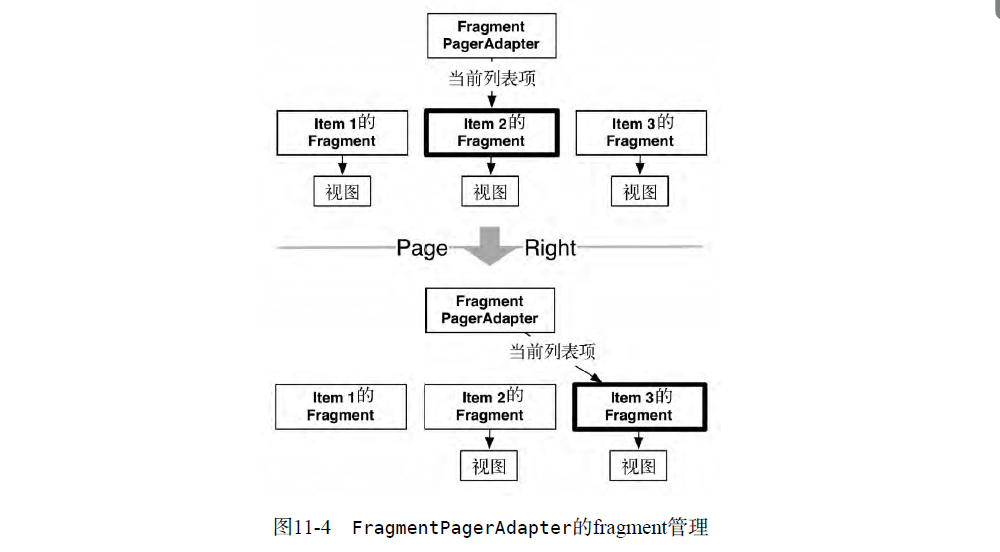
比Activity多了一些生命周期，完整和Activity对接上，大家好好利用。

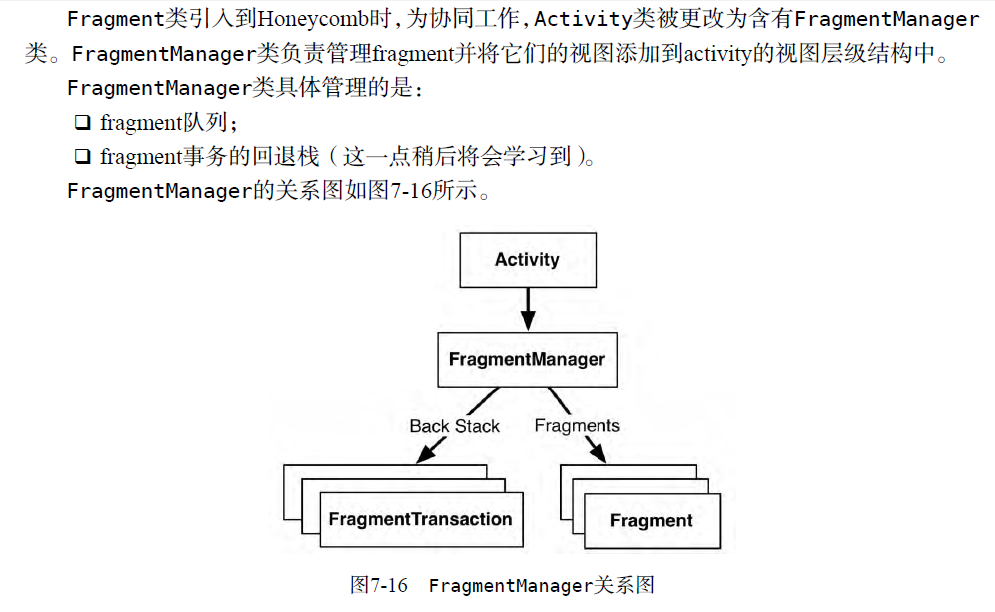




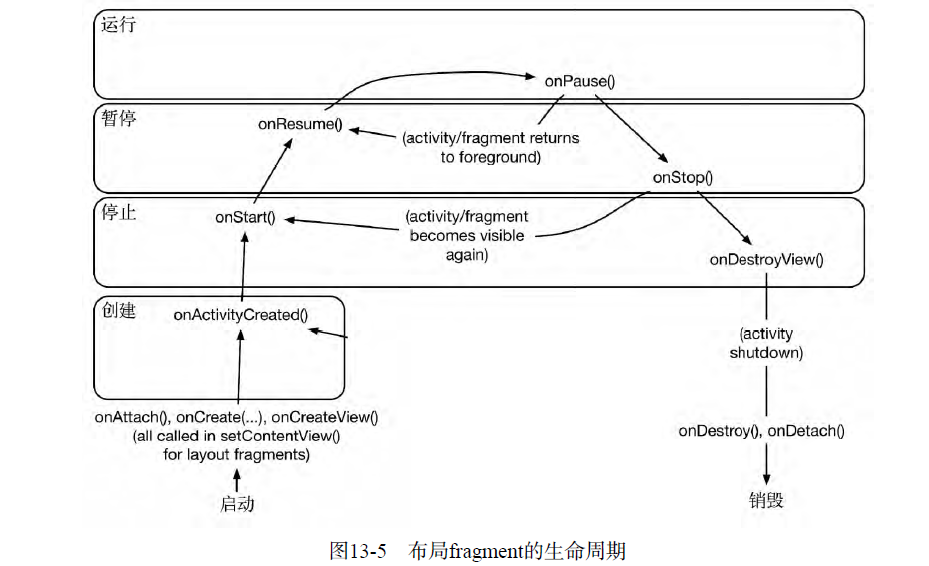










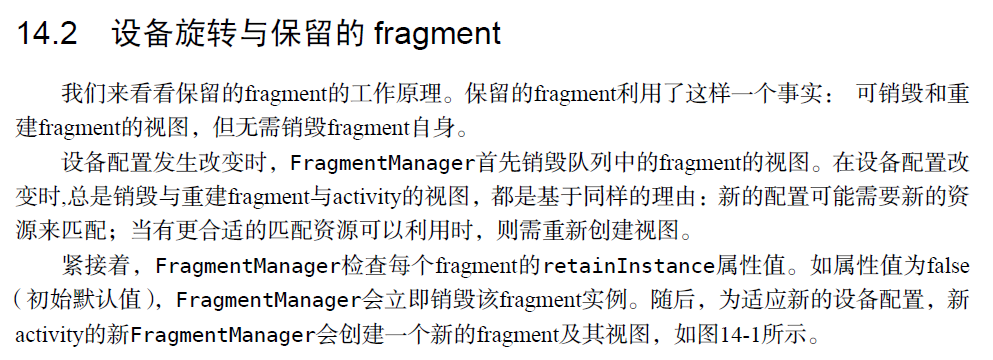


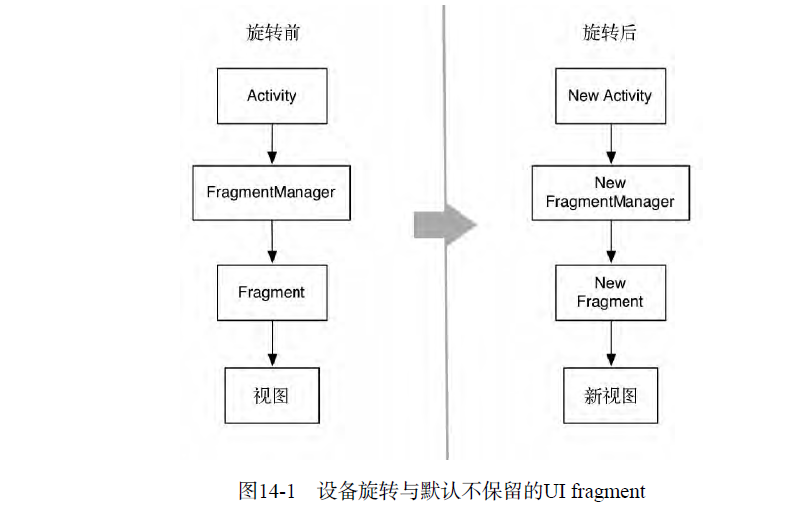
<https://developer.android.com/guide/topics/media/mediaplayer.html>

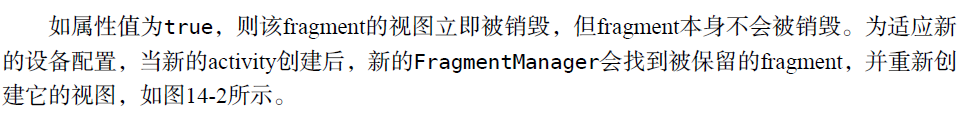
<https://developer.android.com/about/versions/android-4.3.html>

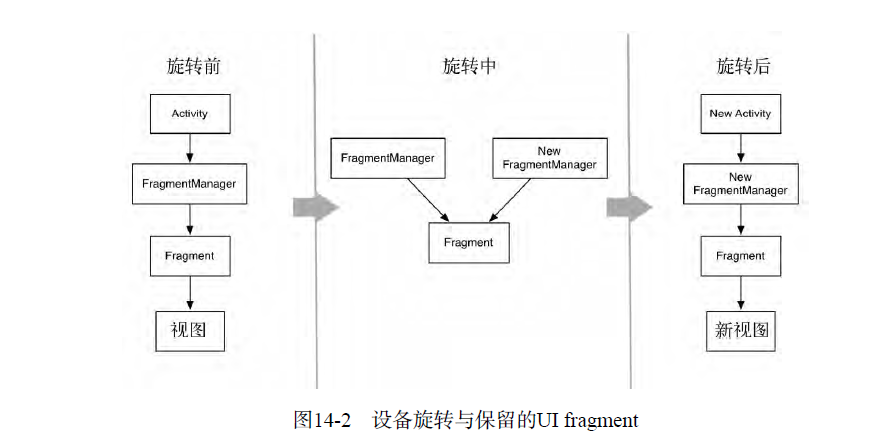
<https://www.locovpn.co/member/pay.php>   
  
Loco加速器翻墙工具: <https://www.locojsq.net/> (1.USER: [262868609@qq.com，登录密码](mailto:262868609@qq.com，登录密码) 14@7B7D@101)

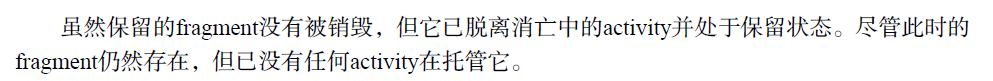
**为庆祝LoCo即将全面覆盖Windows、MAC、IOS、安卓客户端，   
购买代理上GOOGLE等网站学习。**

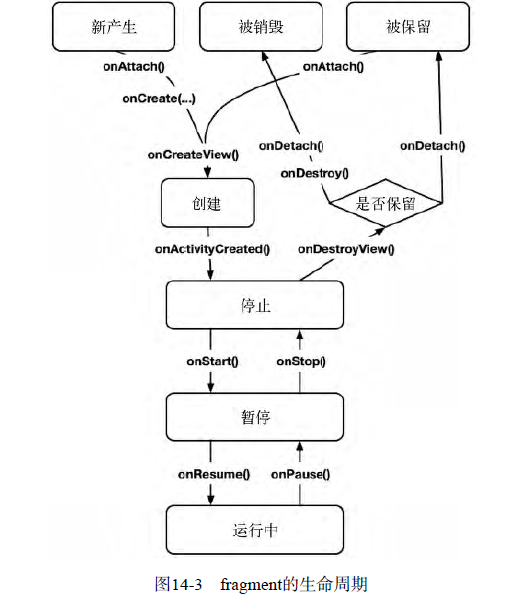


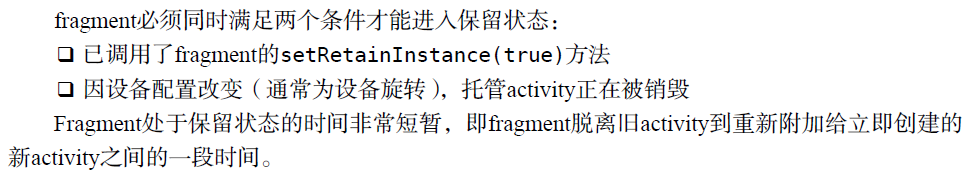




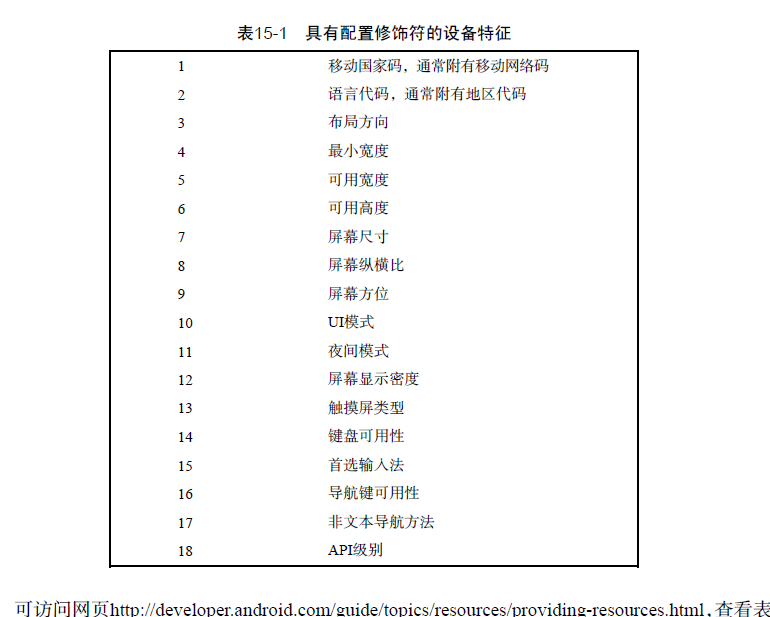


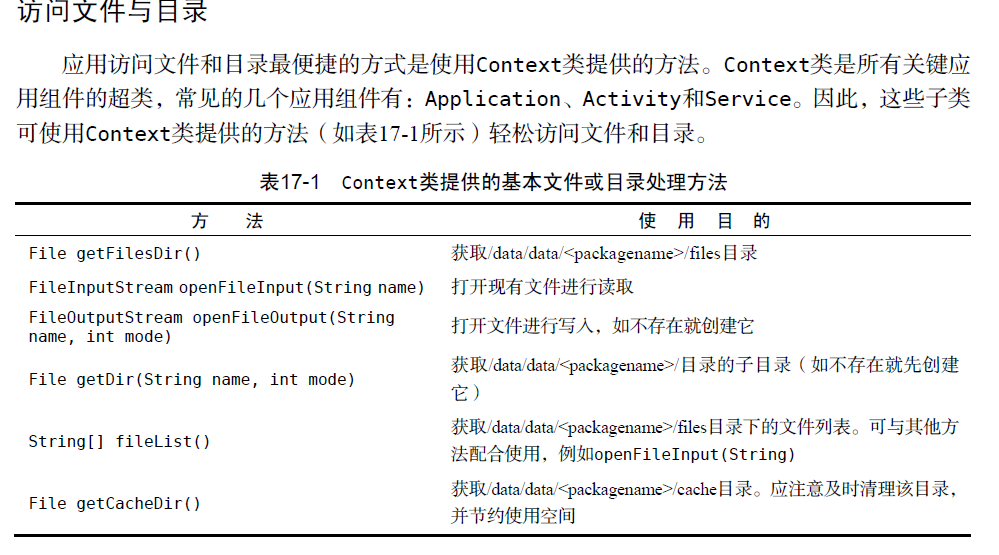


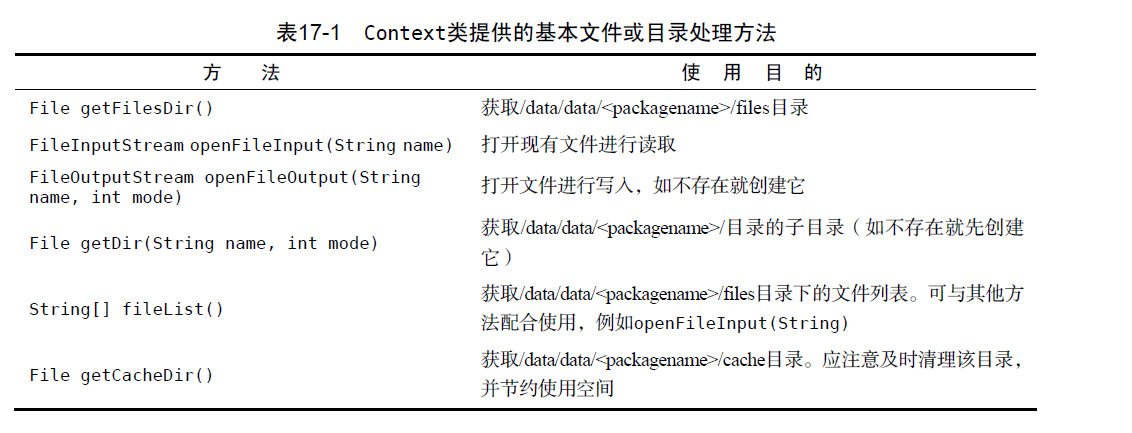




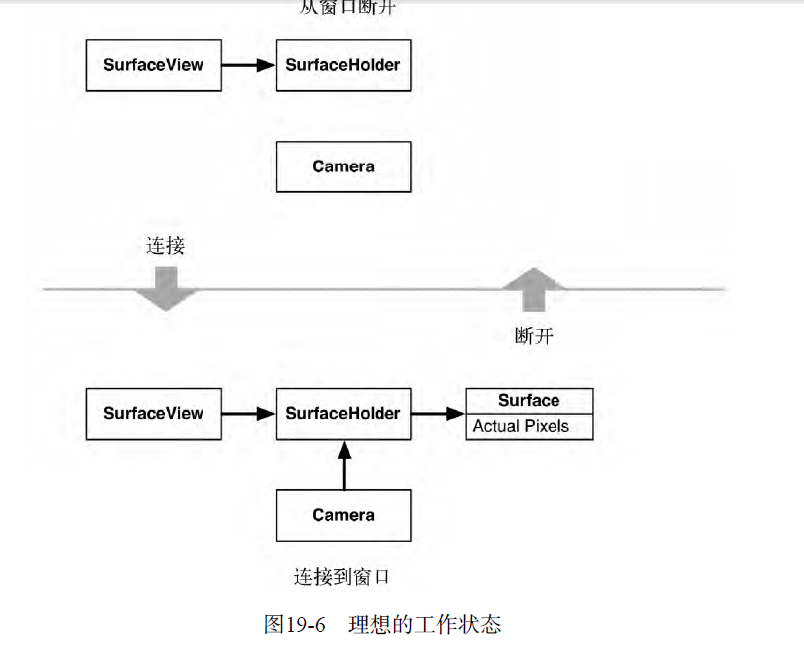
## 15.2 配置修饰符







## 相机的使用



# Android开发实践：掌握Camera的预览方向和拍照方向

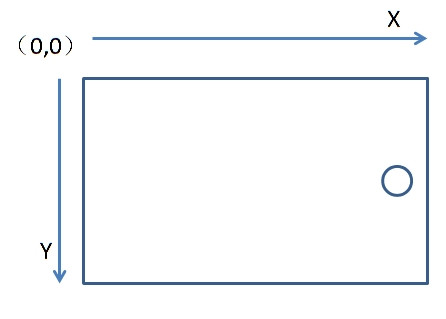
时间 2014-12-21 12:26:25 [对影成三人博客](http://www.tuicool.com/sites/U3y6Vv)

*原文*  [http://ticktick.blog.51cto.com/823160/1592267](http://ticktick.blog.51cto.com/823160/1592267?utm_source=tuicool&utm_medium=referral)

主题 [数码相机](http://www.tuicool.com/topics/10000050)

Android的Camera相关应用开发中，有一个必须搞清楚的知识点，就是Camera的预览方向和拍照方向，本文就重点讨论一下这个问题。

图像的Sensor方向：手机Camera的图像数据都是来自于摄像头硬件的图像传感器（Image Sensor），这个Sensor被固定到手机之后是有一个默认的取景方向的，这个方向如下图所示，坐标原点位于手机横放时的左上角：



Camera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预览界面的画面与你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Sensor采集到的数据进行了旋转处理，然后后才送给显示系统，因此，打开Camera应用后，无论怎么旋转手机，你都能看到“正确”的画面，Android系统提供一个API来手动设置Camera的预览方向，叫做setDisplayOrientation，默认情况下，这个值是0，与图像Sensor方向一致，所以对于横屏应用来说，就不需要更改这个Camera预览方向。但是，如果你的应用是竖屏应用，就必须通过这个API将Camera的预览方向旋转90，与手机屏幕方向一致，这样才会得到正确的预览画面。

Camera的拍照方向：当你点击拍照按钮，得到的图片方向不一定与画面中预览的方向一致，这是因为拍摄的照片是将图像Sensor采集到的数据直接存储到SDCard上的，因此，Camera的拍照方向与上述的Camera的图像Sensor方向一致。

为了演示这个问题，我用手机的Camera对同一个场景拍了两张照片，第一张是横着拿手机拍的，第二张是竖着拿手机拍的。然后用在电脑上打开得到的图片（实际场景中的杯子是竖着的），效果如下所示：



由此可见，如果横向拿手机拍照，由于正好与Camera的拍照方向一致，因此得到的照片是“正确”的；而竖着拿手机拍照的话，Camera的图像Sensor依然以上面描述的角度在采集图像并存储到SDCard上，所以得到的图片就是右图这样的，因为竖着拿手机正好与图像Sensor的方向相差了90度。由此，大家应该明白了为什么我们用手机拍出的照片经常需要旋转90度才能看到“正确”的画面了吧？

我想上面的介绍应该已经把这个问题讲清楚了，下面我还想再深入一下，介绍一下设置Camera预览方向的那个API（setDisplayOrientation）。

上面说了，对于横屏应用，不需要额外设置这个方向，但是对于竖屏应用，则需要调用setDisplayOrientation(90)，来保证Camera的预览方向与Activity的方向一致，那么设置了这个函数究竟会不会影响到Camera拍照的结果呢？根据上面的分析，理论上应该是不影响的，因为拍照得到的图片方向是与图像Sensor的方向一致的，当然，我们可以通过Android官方API的注释文档验证一下这个猜想，下面是Camera.setDisplayOrientation的注释文档：

/\*\*

\* Set the clockwise rotation of preview display in degrees. This affects

\* the preview frames and the picture displayed after snapshot. This method

\* is useful for portrait mode applications. Note that preview display of

\* front-facing cameras is flipped horizontally before the rotation, that

\* is, the image is reflected along the central vertical axis of the camera

\* sensor. So the users can see themselves as looking into a mirror.

\*

\* <p>This does not affect the order of byte array passed in {@link

\* PreviewCallback#onPreviewFrame}, JPEG pictures, or recorded videos. This

\* method is not allowed to be called during preview.

*\*/*

**public** **native** **final** **void** setDisplayOrientation(**int** degrees);

重点看这两句话：

This affects the preview frames **and** the picture displayed after snapshot.

This does **not** affect the order of byte array passed **in** {@link

\* PreviewCallback*#onPreviewFrame}, JPEG pictures, or recorded videos.*

由此我们得到验证了，这个API修改的仅仅是Camera的预览方向而已，并不会影响到PreviewCallback回调、生成的JPEG图片和录像文件的方向，这些数据的方向依然会跟图像Sensor的方向一致。

关于Android Camera的预览和拍照方向的介绍就到这里了，希望对Camera应用开发的新手们能有所帮助，有任何疑问欢迎留言或者来信lujun.hust@gmail.com交流。

## 安全级别：

