Android中直播视频技术探究之---基础知识大纲介绍

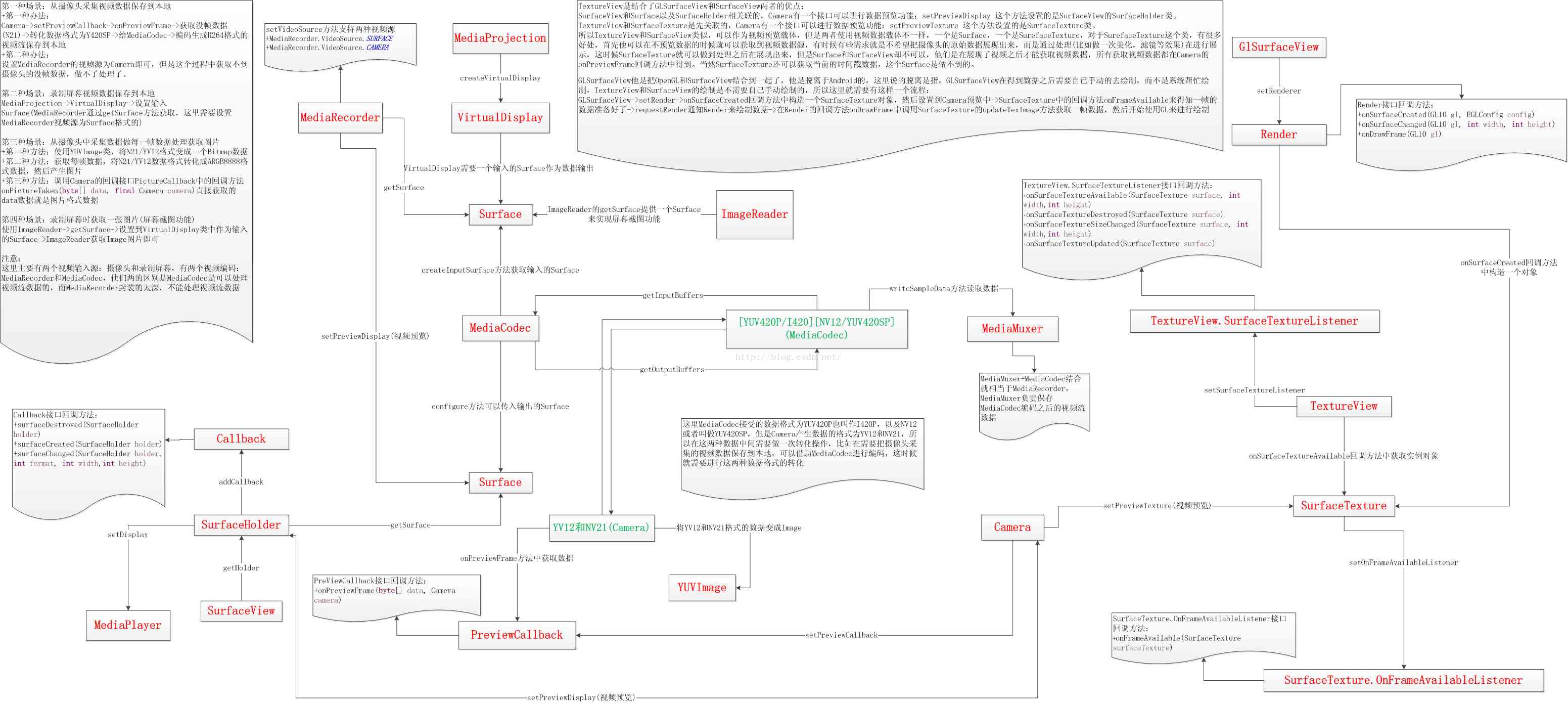
# 一、前言

最近各种视频直播app到处都是，各种霸屏，当然我们也是需要体验的，关于视频直播的软件这里就不介绍了，在不是技术的人来看，直播是一种潮流，是一种娱乐方式，但是作为一个高技术的，我们除了看看，更重要的是学习技术，其实[Android](https://www.2cto.com/kf/yidong/Android/" \t "https://www.2cto.com/kf/201607/_blank)中的视频技术没什么说的，因为网上的资料很多，但是之前的视频技术大部分都出现在了视频播放，就是主流的视频播放器，那个最重要的一个技术就是视频的编解码，这个也会在后续文章中详细介绍视频的处理技术。但是现在直播的技术是在之前的视频技术上又有了一个要求就是视频录制，现在录制很多是借助于牛逼的硬件摄像头。但是除了这个技术，还有其他的我们使用移动设备也可以去解决这个问题。这个后续也会说道如何使用设备去录制视频。

# ****二、知识概要****

从这篇文章开始我们就来介绍一下关于Android中的视频处理，这里主要包括：Android中的摄像头技术，录制视频，视频播放器等知识点，本篇文章是介绍大体的知识点，为后续的章节知识点做铺垫，其实以前在学习Android中最怕就是图片和视频处理方面的技术，因为这些技术有一个基本的要求就是字节操作，考虑很多字节流处理，这个在Android中有一个类ByteBuffer，这个类将会贯穿我们后续所有章节的知识点，这个类也是我们下一篇文章的中点，在视频流处理中这个类将不可或缺的。

下面我们先来看一张Android中视频的处理大纲图解：



# ****三、知识结构****

**第一、视频编码**

Android中视频编码有两种方式，主要是两个核心的类，一个是MediaCodec和MediaRecorder，这两个类有什么区别呢？其实很好理解，他们都可以对视频进行编码，但是唯一不同的是MediaCodec更偏向原生，而MediaRecorder偏向的上层封装。

**1、MediaCodec类**

MediaCodec可以处理具体的视频流，主要有这几个方法：

getInputBuffers：获取需要编码数据的输入流队列，返回的是一个ByteBuffer数组

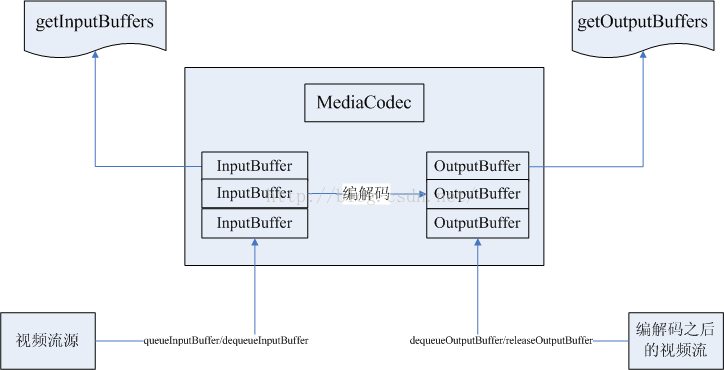
queueInputBuffer：输入流入队列

dequeueInputBuffer：从输入流队列中取数据进行编码操作

getOutputBuffers：获取编解码之后的数据输出流队列，返回的是一个ByteBuffer数组

dequeueOutputBuffer：从输出队列中取出编码操作之后的数据

releaseOutputBuffer：处理完成，释放ByteBuffer数据



这里看到：

视频流有一个输入队列，和输出队列，分别对应getInputBuffers和getOutputBuffers这两个方法获取这个队列，然后对于输入流这端有两个方法一个是queueInputBuffers是将视频流入队列，dequeueInputBuffer是从输入流队列中取出数据进行编解码操作，在输出端这边有一个dequeueOutputBuffer方法从输出队列中获取视频数据，releaseOutputBuffers方法将处理完的输出视频流数据ByteBuffer放回视频流输出队列中，再次循环使用。这样视频流输入端和输出端分别对应一个ByteBuffer队列，这些ByteBuffer可以重复使用，在处理完数据之后再放回去即可。

所以这里看到MediaCodec类处理视频的时候可以接触到视频流数据的，这里比如我们如果有一些特殊需求，比如视频的叠加技术，添加字幕等就可以在这里处理了。同时MediaCodec有一个方法：createInputSurface可以设置视频源输入Surface类型，同时也是可以通过configure方法设置视频输出Surface类型。

**2、MediaRecorder类**

MediaRecorder这个类相对于MediaCodec简单，因为他封装的很好，直接就是几个接口来完成视频录制，比如视频的编码格式，视频的保存路劲，视频来源等，用法简单，但是有一个问题就是不能接触到视频流数据了，处理不了原生的视频数据了。这个也是他和MediaCodec最大的区别，他完成不了视频的叠加技术的。

**注意：**

关于MediaRecorder这个类，其实他和Android中的一个命令是相对应的，就是：adb screenrecord。这个类还有一个地方需要注意的就是他有一个方法：setVideoSource，可以设置视频来源，代码后续文章会介绍，主要就两个来源：一个是来自于设备的摄像头Camera，一个是来自于Surface，关于Surface这个类后面会介绍。

**注意：**

现在视频编码的格式都是H264的，关于H264格式说明如下：

H.264，MPEG-4,MPEG-2等这些都是压缩算法，毕竟带宽是有限的，为了获得更好的图像的传输和显示效果，就不断的想办法去掉一些信息，转换一些信息等等，这就是这些压缩算法的做的事情。H.264最大的优势是具有很高的数据压缩比率，在同等图像质量的条件下，H.264的压缩比是MPEG-2的2倍以上，是MPEG-4的1.5～2倍。举个例子，原始文件的大小如果为88GB，采用MPEG-2压缩标准压缩后变成3.5GB，压缩比为25∶1，而采用H.264压缩标准压缩后变为879MB，从88GB到879MB，H.264的压缩比达到惊人的102∶1！H.264为什么有那么高的压缩比？低码率（Low Bit Rate）起了重要的作用，和MPEG-2和MPEG-4 ASP等压缩技术相比，H.264压缩技术将大大节省用户的下载时间和数据流量收费。尤其值得一提的是，H.264在具有高压缩比的同时还拥有高质量流畅的图像。写了这么多，举个例来说下，比如移动电视，我们接收的到的图像信号一般是H.264格式的，移动设备接收到后，需要先解码成原始的YUV码流，然后又转换成RGB码流，将一帧一帧的RGB数据放到显存上才能显示出图像。虽然传输快了，得是增加了设备的解码成本，不过总体来讲肯定是值得的。现在PC上的显卡慢慢都要集成H.264的硬件解码，据说苹果的最新产品IPAD也是有了这个硬解码。而YUV到RGB的转换，很多ARM芯片上都有了。

**第二、视频数据源**

这里说到的视频数据源就是视频编码器需要编码的视频来源，在移动设备中，我们获取知道两个地方可以获取视频，一个是来自于摄像头Camera，一个是来自于设备屏幕(桌面)。

这里就需要介绍两个类了：一个是摄像头Camera，一个是Android5.0新增的屏幕录制类MediaProjection和VirtualDisplay。

**1、Camear类**

Camera这个类在现在的直播以及美颜相机等app很重要的，他是移动设备采取视频和图片信息的一个重要渠道，他的用法很简单，分为前置摄像头和后置摄像头，可以设置方向，大小等参数，最重要的是，他还需要一个预览界面，他一般预览有两个方法：setPreviewDisplay和setPreviewTexture，第一个方法是设置SurfaceHolder类型的，第二个方法是设置SurfaceTexture类型的关于这两个类型，后面会说到的。但是这里会有一个疑惑了就是来自于摄像头的数据会被预览，但是我们想处理摄像头的数据该怎么办呢？这里就需要借助Camera的一个回调接口：PreviewCallback，这个接口有一个回调方法：onPreviewFrame(byte[] data...)，看到这个方法我们都知道了这个是摄像头采集的视频数据的每一帧数据，我们可以在这里获取每一帧数据然后进行处理，像现在的美颜相机，就是在这里获取到一帧数据，然后在做滤镜效果，然后产生一张图片即可。当然这里可以录制美白视频也是可以的哦。那么在这里我们就可以获取到视频的数据源了。

那么上面的MediaCodec类可以使用getInputBuffer类获取视频流输入队列，我们可以在这个回调方法中获取到数据，然后传入到这个队列中，进行编码操作，但是需要注意的是数据格式需要做一次转化，后面会介绍到。同时MediaRecorder类可以通过setVideoSource方法直接设置视频源。

**2、MediaProjection类和VirtualDisplay类**

这两个类主要是Android5.0新增的一个api，就是专门用来录制设备视频的，不过在使用的过程中需要权限授权的，如果不授权还是很危险的，假如有恶意的软件在后台偷偷的录制设备屏幕视频，就知道你干了啥，那是很危险的。需要通过MediaProjection这个类来获取VirtualDiaplay类，同时需要传入一个重要的参数，就是录制屏幕视频预览的Surface类。

那么这里就可以和上面的视频编码器联系到一起了，MediaCodec有一个createInputSurface方法可以设置视频源类型的Surface，而且MediaRecoder这个类也是可以通过setVideoSource方法设置Surface类型的视频输入源的，在这里如果想实现设备屏幕录制视频可以通过上面的两个视频编码类进行操作然后保存即可。

**第三、视频数据格式**

我们上面看到了两种视频源，一个来自于摄像头，一个来自于屏幕，但是这两个数据源都有自己的格式，所以这里还需要介绍一下数据格式，以及他们之间的转化。我们平常接触的一般都是ARGB颜色空间，A代表透明度，RGB是三原色，但是在处理视频的时候特别是在录制移动设备的时候视频有一个颜色空间：YUV

它也是一种颜色空间，为什么要出现YUV,主要有两个原因，一个是为了让彩色信号兼容黑白电视机，另外一个原因是为了减少传输的带宽。YUV中，Y表示亮度，U和V表示色度，总之它是将RGB信号进行了一种处理，根据人对亮度更敏感些，增加亮度的信号，减少颜色的信号，以这样“欺骗”人的眼睛的手段来节省空间。YUV的格式也很多，不过常见的就是422和420格式。在一般的技术开发中，常用的还是yCbCr,这是一种420格式，也称作I420，注意这个YV12的数据排列刚好是相反的。

Y，U，V它们之间是有一个比例，这个比例不是唯一的，比如Y,U,V三个分量的数量比是4:1:1.也就是说每四个像素共用一对UV。如果是一个30\*40的帧，那么有1200个Y分量，分别有300个U和300个V分量。总共有1200\*1.5这么多个值。

**1、N21/YV12**

这个格式一般是设备的摄像头Camera采集的数据，就是我们上面说到的onPreviewFrame(byte[] data...)每一帧数据，其实是N21或者是YV12格式的，具体哪种格式，可以设置的。所以这里比如我们想获取一帧数据进行处理，一定要记得格式的转化，比如这里想保存一张图片，那么这里就需要将NV21转化成RGB格式的，或者直接使用[系统](https://www.2cto.com/os/" \t "https://www.2cto.com/kf/201607/_blank)类YUVImage，产生一张图片。

**2、YUV420P(I420)/YUV420SP(N12)**

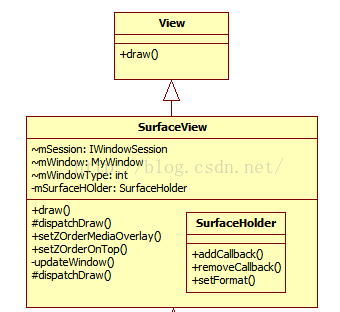
YUV420有打包格式(Packed)，同时还有平面格式(Planar)，即Y、U、V是分开存储的，每个分量占一块地方，其中Y为width\*height，而U、V合占Y的一半，该种格式每个像素占12比特。根据U、V的顺序，分出2种格式，U前V后即YUV420P，也叫I420，V前U后，叫YV12(YV表示Y后面跟着V，12表示12bit)。另外，还有一种半平面格式(Semi-planar)，即Y单独占一块地方，但其后U、V又紧挨着排在一起，根据U、V的顺序，又有2种，U前V后叫NV12，在国内好像很多人叫它为YUV420SP格式；V前U后叫NV21。这种格式似乎比NV16稍受欢迎。

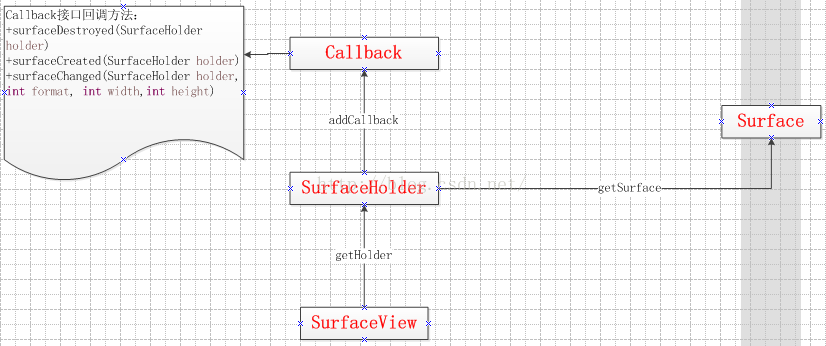
这种格式一般是录制屏幕视频源的格式，就是上面的MediaProjection类，所以我们上面提到的一个将录制设备屏幕视频然后进行编码保存的话，就需要把摄像头的N21/YV12格式转化成编码器识别的YUV420P/YUV420SP格式的。

**第四、视频预览画面**

**1、SurfaceView类**

这个类，我们在开发应用的时候可能会用到的很少，在开发游戏中会用到一些，我们在开发应用制作特殊动画的时候只需要继承View类，然后在onDraw中开始绘制就好了，这个类也可以做到绘制功能，上面看到视频源需要一个预览功能，需要一个Surface，其实SurfaceView就是View+Surface结合体，Surface是图像绘制数据层，而View只是一个展现层，中间还有一个SurfaceHolder作为链接着，他们的关系如下：

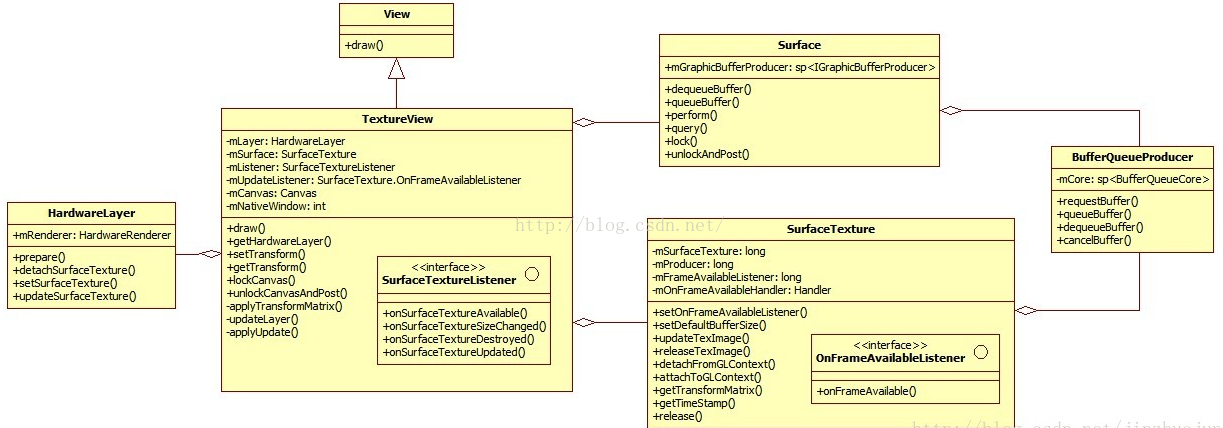


可以通过SurfaceHolder的getSurface方法获取一个Surface类即可。

所以这里的我们在使用SurfaceView作为一个视频预览界面的时候，其实是获取到Surface或者是SurfaceHolder即可，比如摄像头预览界面可以通过Camera的setPreviewDisplay方法设置SurfaceHolder类型即可，屏幕录制界面可以通过VirtualDisplay类参数传递一个输入Surface类型。

**2、TextureView类**

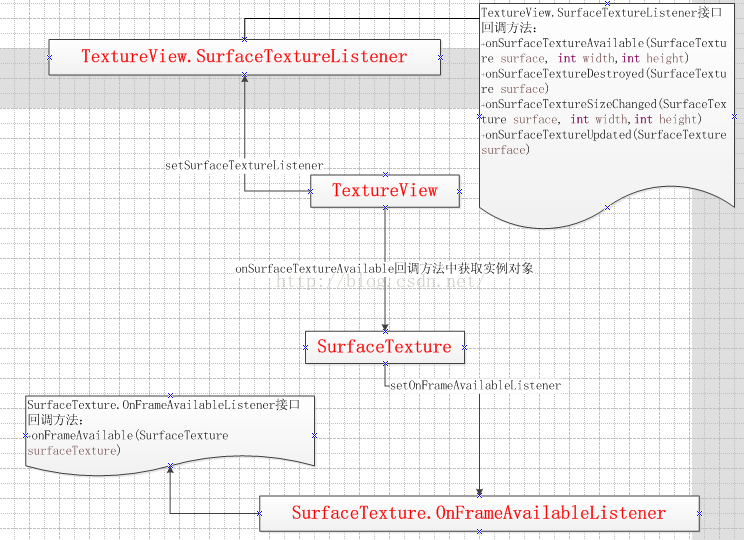
TextureView在4.0(API level 14)中引入。它可以将内容流直接投影到View中，可以用于实现Live preview等功能。和SurfaceView不同，它不会在WMS中单独创建窗口，而是作为View hierachy中的一个普通View，因此可以和其它普通View一样进行移动，旋转，缩放，动画等变化。值得注意的是TextureView必须在硬件加速的窗口中。它显示的内容流数据可以来自App进程或是远端进程。从类图中可以看到，TextureView继承自View，它与其它的View一样在View hierachy中管理与绘制。TextureView重载了draw()方法，其中主要把SurfaceTexture中收到的图像数据作为纹理更新到对应的HardwareLayer中。SurfaceTexture.OnFrameAvailableListener用于通知TextureView内容流有新图像到来。SurfaceTextureListener接口用于让TextureView的使用者知道SurfaceTexture已准备好，这样就可以把SurfaceTexture交给相应的内容源。Surface为BufferQueue的Producer接口实现类，使生产者可以通过它的软件或硬件渲染接口为SurfaceTexture内部的BufferQueue提供graphic buffer。



这个类其实和SurfaceView差不多，只是他内部不是依赖于Surface和SurfacHolder了，而是SurfaceTexture，关于SurfaceTexture它的好处就很多了：

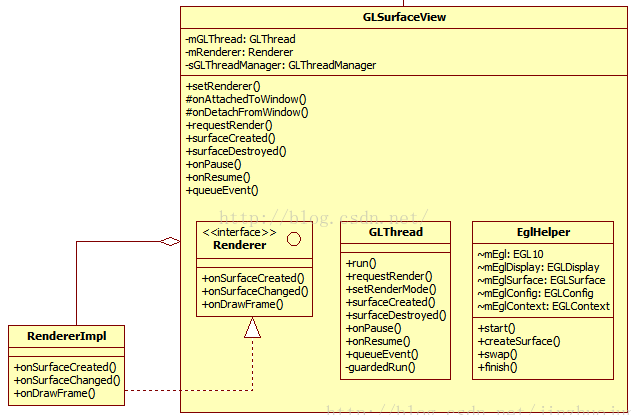
SurfaceTexture是从Android3.0（API 11）加入的一个新类。这个类跟SurfaceView很像，可以从camera preview或者video decode里面获取图像流（image stream）。但是，和SurfaceView不同的是，SurfaceTexture在接收图像流之后，不需要显示出来。有做过Android camera开发的人都知道，比较头疼的一个问题就是，从camera读取到的预览（preview）图像流一定要输出到一个可见的(Visible)SurfaceView上，然后通过Camera.PreviewCallback的onPreviewFrame(byte[] data, Camera camera)函数来获得图像帧数据的拷贝。这就存在一个问题，比如希望隐藏摄像头的预览图像或者对每一帧进行一些处理再显示到手机显示屏上，那么在Android3.0之前是没有办法做到的，或者说你需要用一些小技巧，比如用其他控件把SurfaceView给挡住，注意这个显示原始camera图像流的SurfaceView其实是依然存在的，也就是说被挡住的SurfaceView依然在接收从camera传过来的图像，而且一直按照一定帧率去刷新，这是消耗cpu的，而且如果一些参数设置的不恰当，后面隐藏的SurfaceView有可能会露出来，因此这些小技巧并不是好办法。但是，有了SurfaceTexture之后，就好办多了，因为SurfaceTexture不需要显示到屏幕上，因此我们可以用SurfaceTexture接收来自camera的图像流，然后从SurfaceTexture中取得图像帧的拷贝进行处理，处理完毕后再送给另一个SurfaceView用于显示即可。

而且SurfaceTexture可以轻松的获取视频的时间戳数据，不需要我们人工的去计算，同时他还有一个强大的功能就是和Render结合了，而Render是后面要说到GLSurfaceView的核心，就是OpenGL技术了，对于后续图片和视频的滤镜处理，这个发挥着巨大的作用，因为它对图像流的处理并不直接显示，而是转为GL外部纹理，因此可用于图像流数据的二次处理(如Camera滤镜，桌面特效等)。比如Camera的预览数据，变成纹理后可以交给GLSurfaceView直接显示，也可以通过SurfaceTexture交给TextureView作为View heirachy中的一个硬件加速层来显示。首先，SurfaceTexture从图像流(来自Camera预览，视频解码，GL绘制场景等)中获得帧数据，当调用updateTexImage()时，根据内容流中最近的图像更新SurfaceTexture对应的GL纹理对象，对于Camera数据源的话可以通过setPreviewTexture方法来设置SurfaceTexture类型，录制屏幕数据源的话没有入口可以设置。



**3、GLSurfaceView类**

GLSurfaceView从Android 1.5(API level 3)开始加入，作为SurfaceView的补充。它可以看作是SurfaceView的一种典型使用模式。在SurfaceView的基础上，它加入了EGL的管理，并自带了渲染线程。另外它定义了用户需要实现的Render接口，提供了用Strategy pattern更改具体Render行为的灵活性。作为GLSurfaceView的Client，只需要将实现了渲染函数的Renderer的实现类设置给GLSurfaceView即可。



这里看到GLSurfaceView有一个特点就是不是系统帮我们绘制预览画面了，而是需要我们自己拿到数据之后自己渲染，同时这里会有一个单独的GL线程来进行刷新数据。

# ****四、流程总结****

上面就介绍完了所有类的大致功能以及几种视频源采集的数据格式，下面来总结一下：

**第一、两种编码器MediaCodec和MediaRecorder**

MediaCodec可以通过createInputSurface方法设置输入Surface类型以及configure方法设置输出Surface类型

MediaRecorder可以通过setVideoSource方法设置视频源，两种：一种是摄像头，一种是录制屏幕

这两种编码器的区别在于：MediaCodec可以处理详细的视频流信息，但是MediaRecorder封装太好了，没办法处理。

**第二、两种视频源Camera和MediaProjection**

摄像头数据源提供了一个回调接口中的一个回调方法：onPreviewFrame(byte[] data...)可以获取到视频的每一帧数据

屏幕数据源类VirtualDiaplay提供了一个输入Surface类型的设置入口类型

**第三、视频源格式和视频编码数据格式**

摄像头采集的视频数据格式是N21和YV12，但是编码器MediaCodec处理的数据格式是Y420P和Y420SP的，所以这里需要做一次数据格式的转化，同样如果想采集摄像头的每一帧图片做处理的话，还需要把N21格式转化成RGB格式。

**第四、视频预览View**

1》这里主要有SurfaceView类型，他主要和SurfaceHolder，Surface相关联，摄像头提供了setPreviewDisplay方法设置SurfaceHolder类型。摄像头可以通过SurfaceView进行数据的预览，录制屏幕VirtualDisplay可以提供一个设置Surface入口，所以录制屏幕也可以通过SurfaceView进行数据的预览。

2》还有就是TextureView类型，他主要和SurfaceTexture类相对应的，而摄像头提供了一个setPreviewTexture方法来设置SurfaceTexture类型，但是录制屏幕的VirtualDisplay没有，所以摄像头可以通过TextureView进行数据预览，但是录制屏幕不可以。

3》最后就是GLSurfaceView类型了，他是继承SurfaceView的，在这基础上添加了OpenGL技术，用来处理视频数据和图片数据的。但是GLSurfaceView和前面两个预览View不同的是，他需要拿到数据自己进行渲染预览，大致流程如下：

GLSurfaceView->setRender->onSurfaceCreated回调方法中构造一个SurfaceTexture对象，然后设置到Camera预览中->SurfaceTexture中的回调方法onFrameAvailable来得知一帧的数据准备好了->requestRender通知Render来绘制数据->在Render的回调方法onDrawFrame中调用SurfaceTexture的updateTexImage方法获取一帧数据，然后开始使用GL来进行绘制预览。

# 五、使用场景

**第一种场景：从摄像头采集视频数据保存到本地**  
第一种办法：  
Camera->setPreviewCallback->onPreviewFrame->获取没帧数据(N21)->转化数据格式为Y420SP->给MediaCodec->编码生成H264格式的视频流保存到本地  
第二种办法：  
设置MediaRecorder的视频源为Camera即可，但是这个过程中获取不到摄像头的没帧数据，做不了处理了。  
  
**第二种场景：录制屏幕视频数据保存到本地**  
MediaProjection->VirtualDisplay->设置输入Surface(MediaRecorder通过getSurface方法获取，这里需要设置MediaRecorder视频源为Surface格式的)  
  
**第三种场景：从摄像头中采集数据做每一帧数据处理获取图片**  
第一种方法：使用YUVImage类，将N21/YV12格式变成一个Bitmap数据  
第二种方法：获取每帧数据，将N21/YV12数据格式转化成ARGB8888格式数据，然后产生图片  
第三种方法：调用Camera的回调接口PictureCallback中的回调方法 onPictureTaken(byte[] data, final Camera camera)直接获取的data数据就是图片格式数据  
  
**第四种场景：录制屏幕时获取一张图片(屏幕截图功能)**  
使用ImageReader->getSurface->设置到VirtualDisplay类中作为输入的Surface->ImageReader获取Image图片即可

**第五个场景：对采集的视频和图片做滤镜效果**

我们通过Camera的回调方法onPreviewFrame(byte[] data...)来获取视频流中每一帧数据，然后借助强大的OpenGL技术来进行数据处理，做到类似于美颜相机功能

**第六个场景：扫描二维码技术解析**

可以借助Camera，获取每一帧数据，然后进行二维码的识别。

# ****六、总结****

到这里我们就介绍完了视频直播中大致知识结构，两个数据源，两种编码器，数据格式，三种预览View，但是我们在预览View中看到涉及到了OpenGL技术了，没错，这个也是我们后续需要介绍的内容，如何使用OpenGL做视频的滤镜效果，让每个直播都那么白，后续会详细的结合案例来介绍每个知识点。