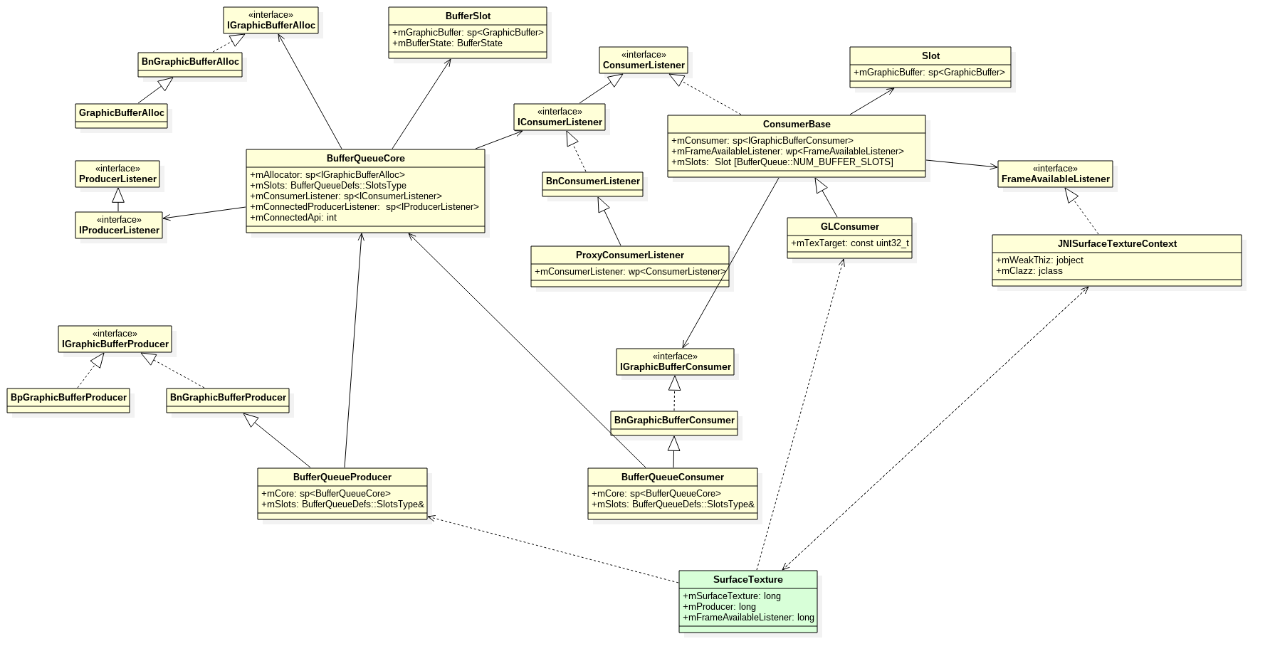
# TODO

# SurfaceTexture

SurfaceTexture，是Android3.0以后引入的，它对图像流的处理并不直接显示，而是转换为GL外部纹理，因此可用于图像流数据的二次处理，比如Camera滤镜、特效等。

提到SurfaceTexture一般会提到与它相关的SurfaceView、GLSurfaceView、TextureView，这几个都可以将图形生产者的数据（比如Camera）送到SurfaceFlinger中显示，而SurfaceTexture可以看做Surface和Texture的组合，是将图形生产者的数据送到Texture，然后是由应用程序自己来处理。这里的Texture应该是属于opengl的概念

## 类图



绿色的为java层的SurfaceTexture对象，黄色的都是native层的对象，可见SurfaceTexture功能基本都在native层实现，java层的SurfaceTexture对象的mSurfaceTexture成员指向的是native层的GLConsumer对象，mProducer成员指向的是native层的BufferQueueProducer对象，当将SurfaceTexture对象传给图形生产者对象比如Camera时，就可以从该成员获取到native层的BufferQueueProducer，用于生产者对象输出

mFrameAvailableListener成员指向的是native层的JNISurfaceTextureContext对象，JNISurfaceTextureContext是OnFrameAvailableListener从native到java的跳板。

该类图与[Android BufferQueue简析](https://www.jianshu.com/p/edd7d264be73)图三的类图很像，那张类图说的是显示流程中的主要类之间的关系，可见SurfaceTexture和SurfaceView原理上基本相似，BufferQueue都是其核心，当然也有一些不同的地方，比如SurfaceTexture的BufferQueue是在应用程序进程这边创建的，而SurfaceView对应的BufferQueue是在SurfaceFlinger进程创建的

### updateTexImage( )

当updateTexImage()方法被调用的时候，SurfaceTexture对象所关联的OpenGL ES中纹理对象的内容将会被更新为image stream中最新的图片，调用该方法可能会丢失帧数据

SurfaceTexture对象可以再任何线程中创建，但是updateTexImage() 方法只能够在OpenGL Thread中被调用，一般情况下都是在onDrawFrame方法中调用updateTexImage() ，将数据绑定给OpenGL 对应的纹理对象。

这里需要注意下，必须显示的调用这个方法，将最新的数据更新到OpenGL ES对应的纹理对象当中去，这样SurfaceTexture才有空间来获取下一帧的数据，不然就永远取不了下一帧数据。

*\* when the SurfaceTexture was created are updated to contain the most recent image from the image  
\* stream. This may cause some frames of the stream to be skipped.*

### getTransformMatrix

当从OpenGL ES纹理取样的时候，需要先调用getTransformMatrix()方法获取变换矩阵，来转换纹理坐标。

每次调用 updateTexImage()这个方法的时候，纹理坐标都有可能变换，所以每次texture image变化的时候，都需要获取一次新的变换矩阵，也就是每调用一次updateTexImage(），也必须要调用一次getTransformMatrix()。

### SurfaceTexture.OnFrameAvailableListener

Callback interface for being notified that a new stream frame is available.

当有新数据可用的时候，调用该方法

当设置渲染器的模式为按需渲染的时候，可以在这个接口回调里面调用GLSurfaceView.requestRender()来进行渲染

具体的实例应用都可以去看看我这篇使用OpenGL显示摄像头

---------------------

作者：Sunshiny\_Lyq

来源：CSDN

原文：https://blog.csdn.net/YuQing\_Cat/article/details/83654295

版权声明：本文为博主原创文章，转载请附上博文链接！

小结

首先，SurfaceTexture从图像流（来自Camera预览、视频解码，GL绘制场景）中获取帧数据，然后使用updateTexImage()将最新的图像更新到SurfaceTexture所对应的GL纹理对象中去，由于SurfaceTexture很特殊，所以对应的OpenGL的纹理对象类型不是普通的sampler2D，而是GL\_TEXTURE\_EXTERNAL\_OES。所以在OpenGL中对纹理进行操作的时候，采样器需要采用额外类型的采样器，另外我们需要通过getTransformMatrix()获取变换矩阵，通过矩阵变换获取到采样点真正的采样位置。

---------------------

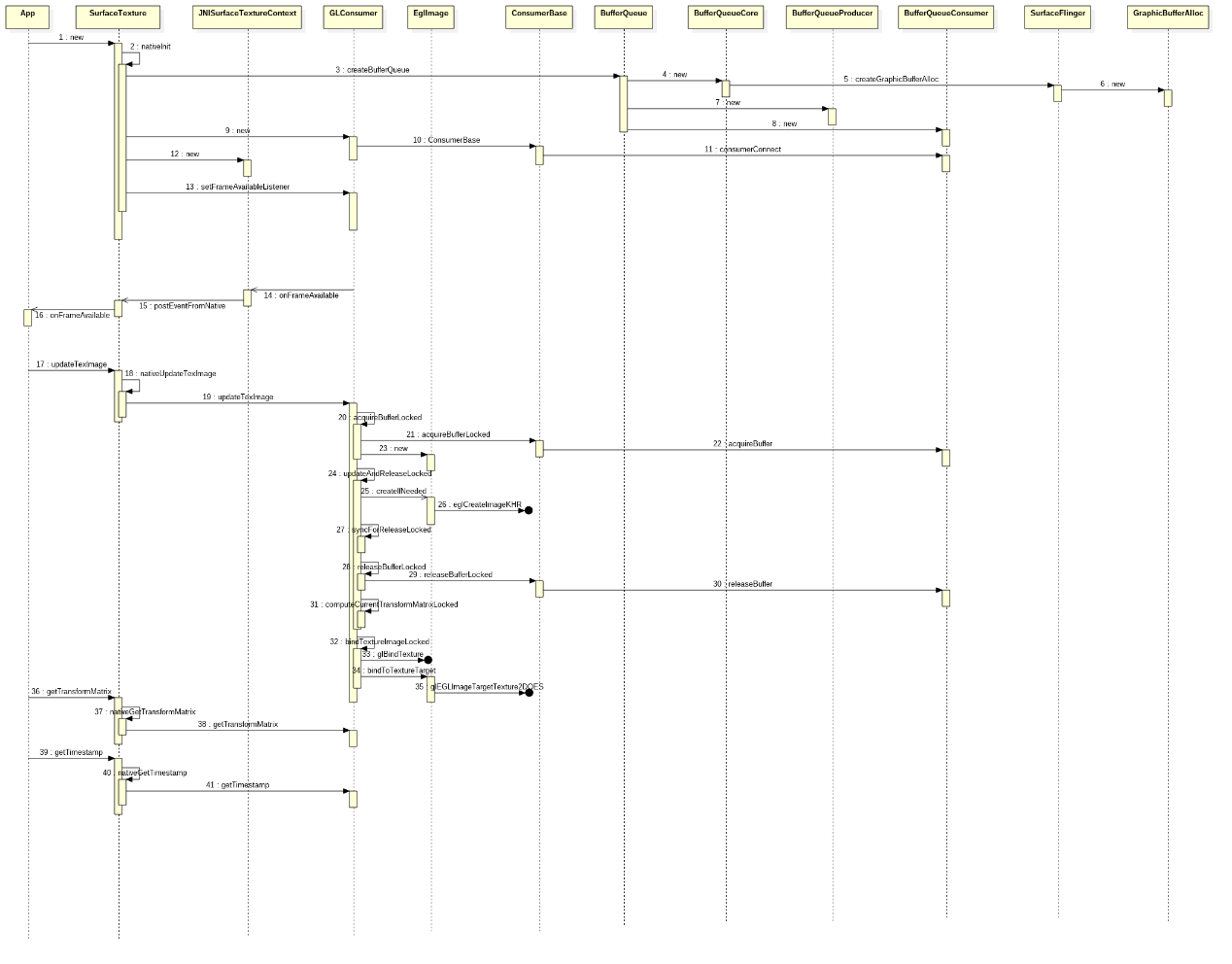
作者：Sunshiny\_Lyq

来源：CSDN

原文：https://blog.csdn.net/YuQing\_Cat/article/details/83654295

版权声明：本文为博主原创文章，转载请附上博文链接！简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。

## 调用流程



应用程序会先创建一个SurfaceTexture，然后将SurfaceTexture传递给图形生产者对象（比如Camera，通过调用setPreviewTexture传递），图形生产者对象生产一帧数据后，会回调onFrameAvailable通知应用程序有新的图像数据可以使用，应用程序就可以调用updateTexImage将图像数据先送到Texture，之后就可以调用opengl接口做些具体的业务了

## step1到step13

这几个流程就是初始化SurfaceTexture，在native层创建了BufferQueue、GLConsumer、JNISurfaceTextureContext，BufferQueue提供图形生产者消费者机制，具体内容请见[Android BufferQueue简析](https://www.jianshu.com/p/edd7d264be73" \t "_blank)，GLConsumer就是SurfaceTexture的图形消费者，即通过调用opengl接口将图形生产者的图像数据送到Texture，JNISurfaceTextureContext是个简单的代理对象，持有java层的SurfaceTexture对象，完成帧可用事件回调。

## step14到step16

这几个流程就是图形生产者生产一帧数据后，通过BufferQueueCore层层调用，最后回调OnFrameAvailableListener的onFrameAvailable，通知Listener有新的图像数据可以使用，一般应用程序会实现该Listener以接收通知。

## step17到step35

这几个流程就是从BufferQueueCore取出图形生产者生产的帧数据GraphicBuffer，然后调用opengl接口将GraphicBuffer更新到Texture上，等熟悉了opengl再来分析。

## step36到step36

应用程序一般还会根据需要调用getTransformMatrix、getTimestamp获取矩阵和时间戳信息，这些信息是在调用updateTexImage时更新的。

通过以上分析，可见要理解SurfaceTexture的关键是要理解BufferQueue和opengl，BufferQueue已经分析过了，opengl等后续熟悉了再抽空分析吧。

作者：Jimmy2012  
链接：https://www.jianshu.com/p/b19a0f49d4ac  
来源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。

作者：Jimmy2012  
链接：https://www.jianshu.com/p/b19a0f49d4ac  
来源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。

## API 注释

从图像流中捕获帧数据用作OpenGL ES的纹理。其中这个图像流数据也可以是相机预览数据也可以是视频解码数据。

当输出目标指定为Camera或者MediaPlayer对象时候，可以使用SurfaceTexture来代替SurfaceHolder如果这样做的话，image stream将把所有帧对象都传给了SurfaceTexture对象，而不是显示在设备上。

## 参考

<https://blog.csdn.net/YuQing_Cat/article/details/83654295>

# SurfaceView

SurfaceView源码分析以及使用

https://blog.csdn.net/YuQing\_Cat/article/details/83625426