SurfaceView渲染解析

1、概览

1.1、Surface

- Definition： surface是一个接口，供生产方和消耗方交换缓冲区。 这里的生产方一般为CPU或者GPU， 消费方为surfaceFlinger。
- surface渲染方式：

  1. lockCanvas() ： 该函数返回canvas，并在cpu上进行渲染。
  2. lockHardwareCanvas(): 返回canvas，并在GPU上渲染。
  3. unLockCanvasAndPost(): 发布到缓冲区，后供surfaceFlinger消费。

  以上三个方法为surfaceHolder提供的方法。

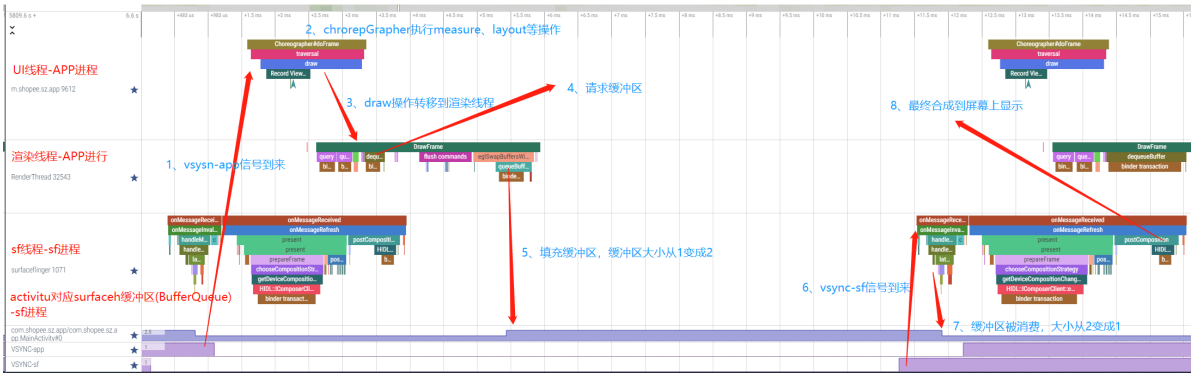
- SurfaceHolder

  每个surfaceView包含一个surfaceHolder，我们对suface的操作首先要获取surfaceView中的surfaceHolder然后使用上面提到的获取画布的方法对surface进行渲染。 但是除了上面提到的获取画布的方式，一些其他的API，如用于播放视频的MediaCodeC可以直接在surface上运行。

Android的View系统中，所有的View对象最终都会渲染到一个surface上面，一般一个window会对应有一个surface。简单回顾下androidView的渲染流程

1. ViewRootImple.scheduleTraversals()被触发(requestLayout、invalidate等方法都可触发)， 消息队列打开同步屏障，往ChoreoGrapher中post mTraversalRunnable(包含measure、layout、draw等操作)， 请求监听Vsyn信号。
2. ChoreoGrapher接收到Vsync信号，执行input、animation、view渲染操作，其中view渲染操作为上面由viewRootImple post的mTraversalRunnable， 里面会执行绘制三连。
3. measure与layout会在CPU中进行，并且会在主线程中进行。 对于draw操作，如果未开启硬件加速，主线程、CPU中进行；如若开启了硬件加速，draw操作会被转接到RenderThread中进行，并使用GPU进行渲染。
4. draw操作开始前会请求Bufferqueue 缓冲区，在渲染结束后提交到缓冲区。 SurfaceFlinger会将缓冲区的数据消费，提交的屏幕上面

我们通过trace可以看到整个流程，帮助理解：



1.2、SurfaceView

SurfaceView是Android官方提供给我们以View的形式便捷使用Surface的组件，SurfaceView具有View的属性，普通的view只能在主线程上进行渲染，但是surfaceView却可以单独开线程去渲染，这也是surfaceview适合视频、游戏等场景的原因。

2、SurfaceView的使用与生命周期

2.1、SurfaceView简单使用

- surfaceView的创建于普通view的创建没什么区别，各种方式都可以
- 对surfaceView进行选时，先使用lockHardwareCanvas，或者lockCanvas获取canvas，获取canvas后可进行渲染，渲染操作可在主线程或者自己创建的线程。

``

```
private void drawSurface() {
    Thread thread = new Thread(new Runnable() {
        @RequiresApi(api = Build.VERSION_CODES.O)
        @Override
        public void run() {
            //允许在自己开启线程进行渲染
            Log.e(TAG, "draw surface");
            Canvas canvas = surfaceView.getHolder().lockHardwareCanvas();
            Paint paint = new Paint();
            paint.setColor(ContextCompat.getColor(getBaseContext(),
R.color.design_default_color_on_primary));
            canvas.drawLine(0f, 0f, 100, 100, paint);
            surfaceView.getHolder().unlockCanvasAndPost(canvas);
        }
    });
    thread.setName("Awille surface");
    thread.start();
}
```

2.2、SurfaceView中surface的生命周期

surfaceView中的surface只会在可见时创建，当app退到后台时，surface会被销毁，并在重新回到后台时重新创建。

在activity中，假如在布局中就声明了surfaceView，surface的创建时机也会等到activity之后。

surfaceView中surface的生命周期回调可以通过以下方式获得。

``

```
surfaceView.getHolder().addCallback(new SurfaceHolder.Callback2() {
    @Override
    public void surfaceRedrawNeeded(@NonNull SurfaceHolder holder) {
        Log.e(TAG, "surfaceRedrawNeeded");
    }
```

```
6
7       @Override
8       public void surfaceCreated(@NonNull SurfaceHolder holder) {
9           Log.e(TAG, "surfaceCreated");
10      }
11
12      @Override
13      public void surfaceChanged(@NonNull SurfaceHolder holder, int format,
    int width, int height) {
14          Log.e(TAG, "surfaceChanged");
15      }
16
17      @Override
18      public void surfaceDestroyed(@NonNull SurfaceHolder holder) {
19          Log.e(TAG, "surfaceDestroyed");
20      }
21  });
```

大家可以自己写个小demo验证一下：

surface生成时机：

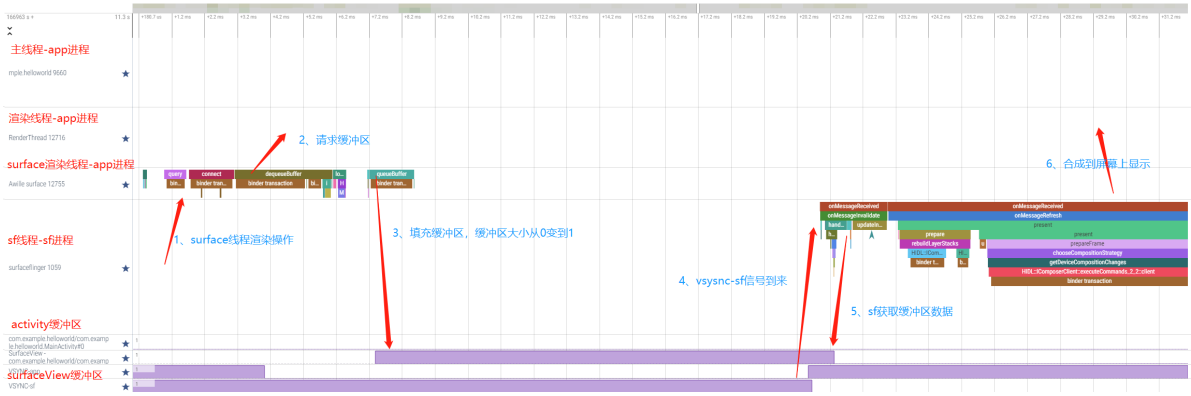
> 2022-09-18 00:04:20.161 10405-10405/com.example.helloworld D/MainActivity: >>>> onStart()
> 2022-09-18 00:04:20.163 10405-10405/com.example.helloworld D/MainActivity: >>>> onResume()
> 2022-09-18 00:04:20.175 10405-10405/com.example.helloworld E/MainActivity: surfaceCreated
> 2022-09-18 00:04:20.175 10405-10405/com.example.helloworld E/MainActivity: surfaceChanged
> 2022-09-18 00:04:20.177 10405-10405/com.example.helloworld E/MainActivity: surfaceRedrawNeeded

surface销毁时机：

> 2022-09-18 00:04:17.601 10405-10405/com.example.helloworld D/MainActivity: >>>> onPause()
> 2022-09-18 00:04:17.608 10405-10405/com.example.helloworld E/MainActivity: surfaceDestroyed
> 2022-09-18 00:04:17.647 10405-10405/com.example.helloworld D/MainActivity: >>>> onStop()

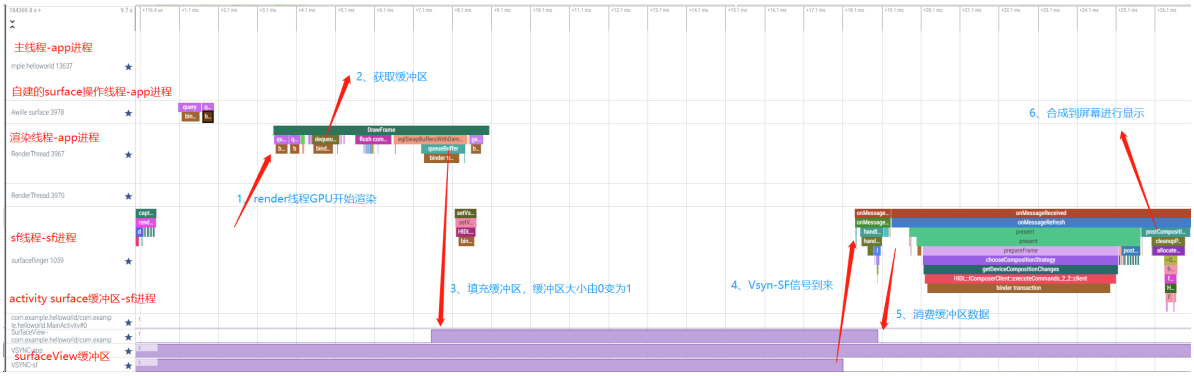
3、SurfaceView渲染解析

3.1、CPU上渲染-lockCanvas



通过trace可以看到，渲染操作在我们开启的线程中进行，并且是使用CPU进行渲染。

3.2、GPU上渲染-lockHardwareCanvas



通过对比别可以发现，lockHardwareCanvas对Canvas进行操作时，是嫁接给RenderThread线程进行渲染的，我们建立的渲染线程只做了一些通知类的操作。

通过trace还有一点发现，获取suface.getHolder获取canvas后，直接对canvas操作，最后post，会直接触发CPU或者GPU直接开始渲染，并没有等到vsync-app信号到来。

3.3、结论

- SurfaceView通过holder获取surface后，是可以使用子线程进行绘制的(通过canvas操作，最终绘制到surface上)。
- 通过lockCanvas对surface进行操作时，渲染操作在CPU中进行，并直接写到缓冲区中，等待Vsync-sf信号到来，surfaceFlinger进行消费。
- 通过lockHardwareCanvas进行操作是，真正的渲染操作会嫁接RenderThread中使用GPU进行，渲染结束后提交到缓冲区，等待Vsync-sf信号到来，surfaceFlinger进行消费。
- 触发渲染的操作不用等待Vsync-app信号触发。(这一点后续增加trace再验证下，目前简单来看，应该是该结论)