surfaceView

# 概述

**SurfaceView**是Android中比较特殊的一类视图，它与普通的View最主要的区别是它和它的视图容器并不是在一个视图层。

Surface的意思是表层，表面的意思，那么SurfaceView就是指一个在表层的View对象。为什么 说是在表层呢，这是因为它有点特殊跟其他View不一样，其他View是绘制在表层外，而它就是充当表层对象。假设你要在一个球上画画，那么球的表层就当 做你的画布对象，你画的东西会挡住它的表层，我们默认没使用SurfaceView，那么球的表层就是空白的，如果我们使用了SurfaceView，我 们可以理解为我们拿来的球本身表面就具有纹路，你是画再纹路之上的，如果你画的是半透明的，那么你将可以透过你画的东西看到球面本身的纹路。SDK的文档 说到：SurfaceView就是在Window上挖一个洞，它就是显示在这个洞里，其他的View是显示在Window上，所以View可以显式在 SurfaceView之上，你也可以添加一些层在SurfaceView之上。

## 为什么要使用SurfaceView

我们知道Android系统中是提供了View来进行绘图处理，然后可以通过invalidate方法通知系统去调用view.onDraw方法去对界面进行重绘，而Android系统是通过发出VSYNC信号来进行屏幕的重绘，可以看出View是主动刷新，而且这个刷新时间是16ms,如果在16ms中没有执行完需要的操作，用户就会看着卡顿，如果draw方法中需要处理的逻辑太多，或者是游戏界面，需要频繁刷新复杂的界面，这样就会阻塞主线程，从而造成画面上的卡顿。而SurfaceView，它的绘制是在另外的线程中，不会去阻塞主线程，另外它底层实现了双缓冲机制。

## MVC框架

我们要了解SurfaceView，还必须要了解和它息息相关的其他两个组件：Surface和SurfaceHolder。Surface其实就视图数据，SurfaceHolder我们都知道是个接口，用来进行绘制。而SurfaceView是显示视图并且和用户交互的界面。

而MVC（Model-View-Controller）框架：model数据==Surface，View显示==SurfaceView，控制器==SurfaceHolder。

## Surface源码

public class Surface implements Parcelable {

//code....

}

首先看一下这个Surface类，实现了Parcelable接口进行了序列化（可以在进程中传递该类对象）用来处理屏幕显示缓冲区的数据。

源码中的注释是：Handle onto a raw buffer that is being managed by the screen compositor 翻译一下就是 对由屏幕图像合成器管理的原始缓冲区进行处理，也就是用来获取原始缓冲区以及其中的内容，原始缓冲区（raw buffer）是用来保存当前窗口的像素数据，由此可知Surface就是Android用来绘图的地方，

既然需要绘图，那肯定需要画布，也就是在其内部定义了Canvas对象

private final Canvas mCanvas = new CompatibleCanvas();

CompatibleCanvas是Surface的内部类，它其中包含了一个Matrix对象，这个矩阵本质就是一块内存区域，对View的各种绘图操作就保存在这片内存中。

这个内部类的作用是为了兼容Android各个分辨率的屏幕，根据不同屏幕的分辨率处理不同的图像数据

lockCanvas

这个方法很重要，这个方法其实并不是SurfaceView调用的，是由surfaceHolder调用。API中的解释：

Gets a Canvas for drawing into this surface. ( 获取当前正在绘画的canvas对象 ) After drawing into the provided Canvas, the caller must invoke unlockCanvasAndPost(Canvas) to post the new contents to the surface.(绘制完一帧数据之后，需要调用unlockCanvasAndPost释放画布，然后把绘制好的内容Post到屏幕上去显示)

也就是说在绘画的时候，这个画布是被锁定的，也就是说只有当前的绘制操作完成并且画布解锁以后才能对画布进行其他的操作。

unlockCanvasAndPost

Posts the new contents of the Canvas to the surface and releases the Canvas. 将新绘制的内容传递给Surface，这个canvas就会释放掉

总结

Surface中提供一个可以用来处理不同屏幕分辨率的Canvas，这个Canvas就是用来提供给程序员画画的，在画画的过程当中，需要先锁定画布，只有在完成当前绘制，把数据post给Surface并且解锁画布之后，，才可以继续绘制其他的内容，原始缓冲区（raw buffer）就是用来保存数据的，而Surface的作用就类似于C++当中的句柄，得到这个句柄，就可以得到其中的Canvas、原始缓冲区以及其中内容

SurfaceHolder

android.view.SurfaceHolder 是一个接口

public interface SurfaceHolder {

}

1

2

3

API解释：

Abstract interface to someone holding a display surface.

针对显示界面Surface的抽象接口

Allows you to control the surface size and format, edit the pixels in the surface, and monitor changes to the surface. This interface is typically available through the SurfaceView class.

允许你去控制这个Surface界面的大小、格式以及在上面绘画，并且可以监控Surface的变化，这个接口通常可以通过 SurfaceView 这个类获得（SurfaceView的getHolder()方法）这个的作用就是上面提到的Controller的角色

When using this interface from a thread other than the one running its SurfaceView, you will want to carefully read the methods lockCanvas() and Callback.surfaceCreated().

如果使用子线程来使用这个接口，你需要非常注意 lockCanvas() and Callback.surfaceCreated()这两个方法的使用

关键接口 Callback

它是SurfaceHolder内部的一个接口，有三个方法

public void surfaceCreated(SurfaceHolder holder); Surface第一次被创建时候调用，Surface由不可见状态到可见状态，从这个可见状态开始一直到surfaceDestroyed销毁之前，这段时间Surface对象是可以进行操作的

public void surfaceChanged(SurfaceHolder holder, int format, int width, int height); Surface的大小和格式改变的时候会调用，比如横竖屏切换的时候就会被调用，这个方法在surfaceCreated之后至少会被调用一次

public void surfaceDestroyed(SurfaceHolder holder); Surface被销毁时候调用，注意调用这个方法之后，不能够在对Surface进行操作了，否则会报错

总结

SurfaceHolder是一个接口，提供访问和控制SurfaceView中内嵌的Surface的相关方法。

## DEMO

整个过程：继承SurfaceView并实现SurfaceHolder.Callback接口 ----> SurfaceView.getHolder()获得SurfaceHolder对象 ---->SurfaceHolder.addCallback(callback)添加回调函数---->SurfaceHolder.lockCanvas()获得Canvas对象并锁定画布----> Canvas绘画 ---->SurfaceHolder.unlockCanvasAndPost(Canvas canvas)结束锁定画图，并提交改变，将图形显示。

<lyq.com.surfaceviewdemo.MySurfaceView

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

/>

**public class** MySurfaceView **extends** SurfaceView **implements** Runnable, SurfaceHolder.Callback {  
 **private** SurfaceHolder **mHolder**;  
 *//用于绘图的Canvas* **private** Canvas **mCanvas**;  
 *//子线程标志位* **private boolean mIsDrawing**;  
 *//画笔* **private** Paint **mPaint**;  
 *//路径* **private** Path **mPath**;  
 **public** MySurfaceView(Context context) {  
 **super**(context);  
 initView();  
 }  
 **public** MySurfaceView(Context context, AttributeSet attrs) {  
 **super**(context, attrs);  
 initView();  
 }  
  
 **public** MySurfaceView(Context context, AttributeSet attrs, **int** defStyleAttr) {  
 **super**(context, attrs, defStyleAttr);  
 initView();  
 }  
  
 **private void** initView() {  
 **mHolder** = getHolder();  
 *//添加回调* **mHolder**.addCallback(**this**);  
 **mPath**=**new** Path();  
 *//初始化画笔* **mPaint**=**new** Paint();  
 **mPaint**.setStyle(Paint.Style.STROKE);  
 **mPaint**.setStrokeWidth(6);  
 **mPaint**.setAntiAlias(**true**);  
 **mPaint**.setColor(Color.RED);  
 setFocusable(**true**);  
 setFocusableInTouchMode(**true**);  
 **this**.setKeepScreenOn(**true**);  
 }  
 *//Surface的生命周期* @Override  
 **public void** surfaceCreated(SurfaceHolder holder) {  
 **mIsDrawing**=**true**;  
 **new** Thread(**this**).start();  
 }  
 @Override  
 **public void** surfaceChanged(SurfaceHolder holder, **int** format, **int** width, **int** height) { }  
  
 @Override  
 **public void** surfaceDestroyed(SurfaceHolder holder) {  
 **mIsDrawing**=**false**;  
 }  
  
 @Override  
 **public void** run() {  
 **long** start =System.*currentTimeMillis*();  
 **while**(**mIsDrawing**){  
 draw();  
 **long** end = System.*currentTimeMillis*();  
 **if**(end-start<100){  
 **try**{  
 Thread.*sleep*(100-end+start);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
  
 **private void** draw() {  
 **try**{  
 *//锁定画布并返回画布对象* **mCanvas**=**mHolder**.lockCanvas();  
 *//接下去就是在画布上进行一下draw* **mCanvas**.drawColor(Color.WHITE);  
 **mCanvas**.drawPath(**mPath**,**mPaint**);  
  
 }**catch** (Exception e){  
 }**finally** {  
 *//当画布内容不为空时，才post，避免出现黑屏的情况。* **if**(**mCanvas**!=**null**)  
 **mHolder**.unlockCanvasAndPost(**mCanvas**);  
 }  
 }  
 */\*\*  
 \* 绘制触摸滑动路径  
 \*/* @Override  
 **public boolean** onTouchEvent(MotionEvent event) {  
 **int** x=(**int**) event.getX();  
 **int** y= (**int**) event.getY();  
 **switch** (event.getAction()){  
 **case** MotionEvent.ACTION\_DOWN:  
 Log.*d*(***TAG***, **"onTouchEvent: down"**);  
 **mPath**.moveTo(x,y);  
 **break**;  
 **case** MotionEvent.ACTION\_MOVE:  
 Log.*d*(***TAG***, **"onTouchEvent: move"**);  
 **mPath**.lineTo(x,y);  
 **break**;  
 **case** MotionEvent.ACTION\_UP:  
 Log.*d*(***TAG***, **"onTouchEvent: up"**);  
 **break**;  
 }  
 **return true**;  
 }  
 */\*\*  
 \* 清屏  
 \*/* **public boolean** reDraw(){  
 **mPath**.reset();  
 **return true**;  
 }  
}

# SurfaceView播放视频

## SurfaceView

先来介绍一下大部分软件如何解析一段视频流。首先它需要先确定视频的格式，这个和解码相关，不同的格式视频编码不同，不是这里的重点。知道了视频的编码格式后，再通过编码格式进行解码，最后得到一帧一帧的图像，并把这些图像快速的显示在界面上，即为播放一段视频。SurfaceView在Android中就是完成这个功能的。

　　既然SurfaceView是配合MediaPlayer使用的，MediaPlayer也提供了相应的方法设置SurfaceView显示图片，只需要为MediaPlayer指定SurfaceView显示图像即可。它的完整签名如下：

　　　　void setDisplay(SurfaceHolder sh)

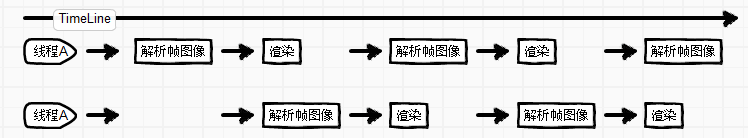
　　它需要传递一个SurfaceHolder对象，SurfaceHolder可以理解为SurfaceView装载需要显示的一帧帧图像的容器，它可以通过SurfaceHolder.getHolder()方法获得。

使用MediaPlayer配合SurfaceView播放视频的步骤与播放使用MediaPlayer播放MP3大体一致，只需要额外设置显示的SurfaceView即可。

## SurfaceView双缓冲

上面有提到，SurfaceView和大部分视频应用一样，把视频流解析成一帧帧的图像进行显示，但是如果把这个解析的过程放到一个线程中完成，可能在上一帧图像已经显示过后，下一帧图像还没有来得及解析，这样会导致画面的不流畅或者声音和视频不同步的问题。所以SurfaceView和大部分视频应用一样，通过双缓冲的机制来显示帧图像。那么什么是双缓冲呢？双缓冲可以理解为有两个线程轮番去解析视频流的帧图像，当一个线程解析完帧图像后，把图像渲染到界面中，同时另一线程开始解析下一帧图像，使得两个线程轮番配合去解析视频流，以达到流畅播放的效果。

下图为演示了双缓冲的过程，线程A和线程B配合解析渲染视频流的帧图像



## SurfaceHolder

SurfaceView内部实现了双缓冲的机制，但是实现这个功能是非常消耗系统内存的。因为移动设备的局限性，Android在设计的时候规定，SurfaceView如果为用户可见的时候，创建SurfaceView的SurfaceHolder用于显示视频流解析的帧图片，如果发现SurfaceView变为用户不可见的时候，则立即销毁SurfaceView的SurfaceHolder，以达到节约系统资源的目的。

　　如果开发人员不对SurfaceHolder进行维护，会出现最小化程序后，再打开应用的时候，视频的声音在继续播放，但是不显示画面了的情况，这就是因为当SurfaceView不被用户可见的时候，之前的SurfaceHolder已经被销毁了，再次进入的时候，界面上的SurfaceHolder已经是新的SurfaceHolder了。所以SurfaceHolder需要我们开发人员去编码维护，维护SurfaceHolder需要用到它的一个回调，SurfaceHolder.Callback()，它需要实现三个如下三个方法：

void surfaceDestroyed(SurfaceHolder holder)：当SurfaceHolder被销毁的时候回调。

void surfaceCreated(SurfaceHolder holder)：当SurfaceHolder被创建的时候回调。

void surfaceChange(SurfaceHolder holder)：当SurfaceHolder的尺寸发生变化的时候被回调。

　　以下是这三个方法的调用的过程，在应用中分别为SurfaceHolder实现了这三个方法，先进入应用，SurfaceHolder被创建，创建好之后会改变SurfaceHolder的大小，然后按Home键回退到桌面销毁SurfaceHolder，最后再进入应用，重新SurfaceHolder并改变其大小。

## SurfaceView的兼容性

对于Android4.0以下的设备，在使用SurfaceView播放视频的时候，需要为其设置一个额外的属性。之前提到过，SurfaceView维护了一个双缓冲的机制，它会自己维护缓冲区，无需我们手动维护

## SurfaceView的Demo示例

上面讲了那么多关于SurfaceView的内容，下面通过一个Demo简单演示一下SurfaceView如何播放视频，加了一个滚动条，用于显示进度，还可以拖动滚动条选择播放位置，Demo的注释比较完整，这里不再累述，视频是在网上随便找的，朋友们运行的时候保证/sdcard/ykzzldx.mp4，这个目录下有这个文件。

https://www.cnblogs.com/plokmju/p/android\_SurfaceView.html