# Android程序设计

SurfaceView和对话框

2019.2.27

isszym　sysu.edu.cn

官方文档（中文） 官方文档（英文） runoob

# 目录

- SurfaceView
- TextureView对话框概述

# SurfaceView画图

概述

SurfaceView

TextureView

参考　参考

# 概述

- Android应用程序只有两种线程，即UI主线程(UI thread)和工作线程(work thread)。手机屏幕每秒60帧就不会卡顿，大约16ms刷新一次。Activity的整个显示过程包括所有View的测量、布局、绘制和计算。这些都是在UI主线程中完成的。如果这个过程的执行大于16ms，就会影响屏幕刷新和响应，出现卡顿现象。因此，计算量大的任务要放到工作线程（子线程）中去完成。
- **SurfaceView**是View的子类。它使用了双缓冲机制，在新的线程中利用一个缓冲区绘制好屏幕，然后提交到UI主界面显示它。由于SurfaceView不会影响主线程的更新速度，它常用于游戏开发和视频播放。
- 由于SurfaceView只能在整个屏幕中而不能作为一个View进行绘制。因此，Android 4.0中引入了TextureView。
- **TextureView**是View的子类，它和SurfaceView一样，可以在独立的线程中绘制和渲染，并利用专用的GPU线程提高渲染的性能。与SurfaceView不同的是TextureView可以看成和Button、TextView一样的普通控件，可以使用平移、缩放、旋转等变换，也可以使用View.setAlpha()等操作。TextureView只能使用在硬件加速开启的窗口中。
- **GLSurfaceView是**SurfaceView的子类，专门负责OpenGL渲染。

# SurfaceView

SurfaceView surface

- SurfaceView是视图(View)的子类，这个视图里内嵌了一个专门用于绘制的Surface。可以定义这个Surface的格式和尺寸，并用SurfaceView控制这个Surface的绘制位置。
- Surface总在自己所在窗口的后面，SurfaceView提供了一个可见区域，只有处于这个可见区域内的Surface内容才可见，可见区域外的部分不可见。Surface的显示受到视图层级(z-index)关系的影响，它显示的内容会被它的上层控件遮挡。如果Surface上面放的是透明控件，那么它的每次变化都会引起透明效果的重新计算，这对性能有一定影响。
- 继承SurfaceView的应用类可以通过getHolder()获得SurfaceHolder，这是Surface的控制器。应用类还必须实现SurfaceHolder.Callback接口。该接口要求定义回调函数surfaceCreated(SurfaceHolder)和surfaceDestroyed(SurfaceHolder)。Surface只在创建和销毁事件之间才存在。
- SurfaceView编程的核心在于提供两个线程：UI线程和渲染线程。所有SurfaceView和SurfaceHolder.Callback的方法都应该在UI线程里调用，一般来说就是应用程序主线程。渲染线程用于Surface的绘制。Surface可以被直接复制到显存从而显示出来。

参考

- SurfaceHolder用于操纵Surface，可以在它的Canvas上作画，它的方法有：
  （1） abstract void addCallback(SurfaceHolder.Callback callback);
  给SurfaceView当前的持有者一个回调对象（当前对象）。
  （2） abstract Canvas lockCanvas();
  锁定画布并返回的画布对象Canvas，然后就可以画图了。
  （3） abstract Canvas lockCanvas(Rect dirty);
  锁定画布的某个区域。不用重画dirty外的其它区域的像素以提高速度。
  （4） abstract void unlockCanvasAndPost(Canvas canvas);
  把canvas的新内容(复制到显存)提交给Surface并释放canvas。

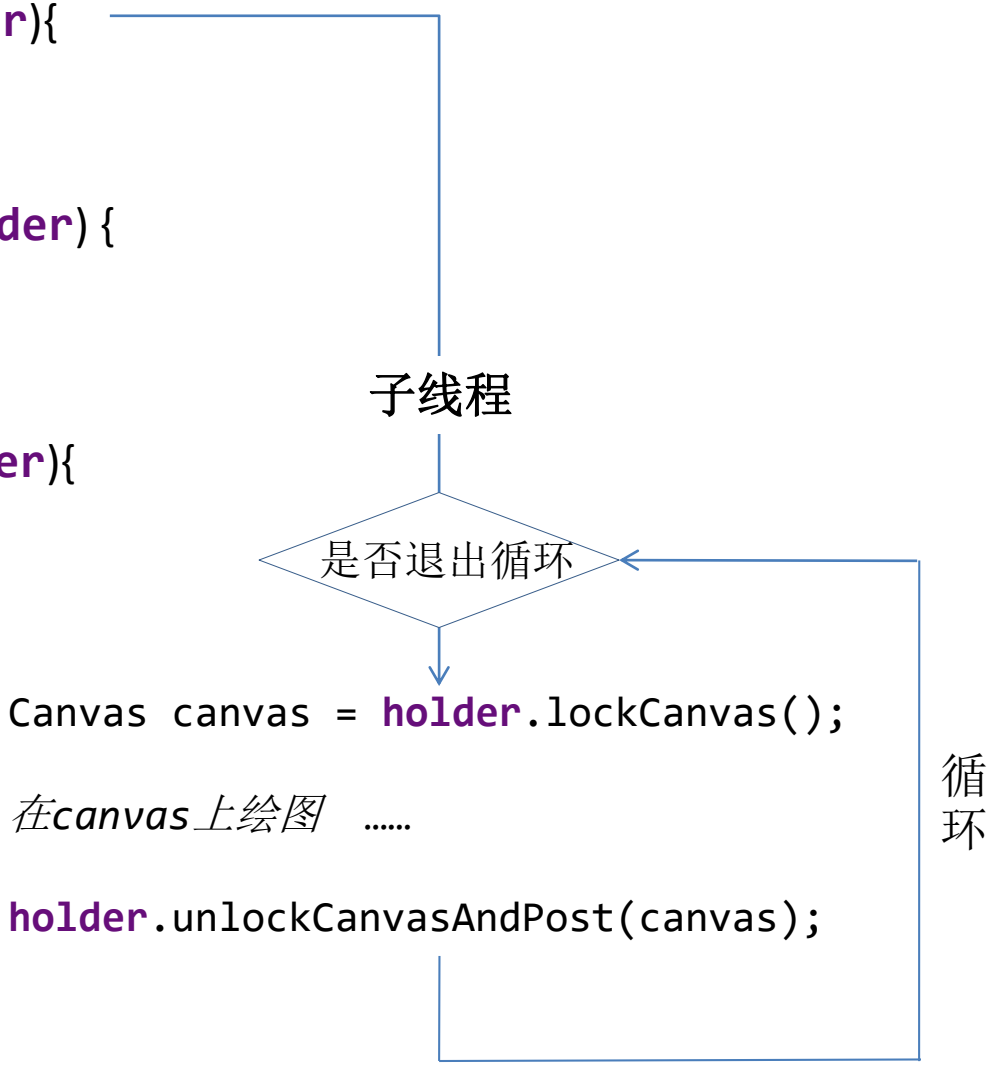
- 应用类需要重写的方法：
  （1）public void surfaceChanged(SurfaceHolder holder,int format,int width,int height){}
  在surface的大小发生改变时激发
  （2）public void surfaceCreated(SurfaceHolder holder){}
  在创建时激发，一般在这里调用画图的线程。
  （3）public void surfaceDestroyed(SurfaceHolder holder) {}
  销毁时激发，一般在这里将画图的线程停止、释放。

## SurfaceView

surfaceCreated(SurfaceHolder **holder**){
　建立子线程
}

surfaceDestroyed(SurfaceHolder **holder**) {
　给出退出循环的条件
}

surfaceChanged(SurfaceHolder **holder**){
　在surface的大小发生改变时激发
}



# 项目名:NewSurfaceView

MyView.java

```
// 实现SurfaceHolder的回调接口
public class MyView extends SurfaceView implements SurfaceHolder.Callback {
    private SurfaceHolder holder;
    private MyThread myThread;
    public MyView(Context context) {
        super(context);
        holder = this.getHolder();
        holder.addCallback(this);     // 给SurfaceView当前的持有者一个回调对象
        myThread = new MyThread(holder); //创建一个绘图线程
    }
    @Override  //在surface的大小发生改变时激发
    public void surfaceChanged(SurfaceHolder holder, int format, int width,
                    int height) {
    }
    @Override  //在创建时激发，可以在这里调用画图的线程。
    public void surfaceCreated(SurfaceHolder holder){
        myThread.isRun = true;
        myThread.start();
    }
    @Override //销毁时激发，一般在这里将画图的线程停止、释放。
    public void surfaceDestroyed(SurfaceHolder holder) {
        // TODO Auto-generated method stub
        myThread.isRun = false;
    }
}
```

```
class MyThread extends Thread{
    private SurfaceHolder holder; public boolean isRun ;
    public  MyThread(SurfaceHolder holder) {
       this.holder =holder;  isRun = true;
    }
    @Override  public void run() {
      int count = 0;
      while(isRun) {
        Canvas c = null;
        try { synchronized (holder) {   //利用holder加同步锁
              c = holder.lockCanvas(); //锁定画布并返回可做作画的对象Canvas
              c.drawColor(Color.BLUE); //设置画布背景颜色
              Paint p = new Paint();   //创建画笔
              p.setColor(Color.WHITE);
              Rect r = new Rect(100, 100, 600, 600);
              c.drawRect(r, p);
              p.setTextSize(60); p.setColor(Color.RED);
              c.drawText("这是第"+(count++)+"秒", 120, 180, p);
              Thread.sleep(1000);   //睡眠1000毫秒
            }
        }
        catch (Exception e) {e.printStackTrace();  }
        finally {
           if(c!= null) {
              holder.unlockCanvasAndPost(c);//结束锁定并提交改变。
          }
        }
      } //while
   }
}
```

# TextureView

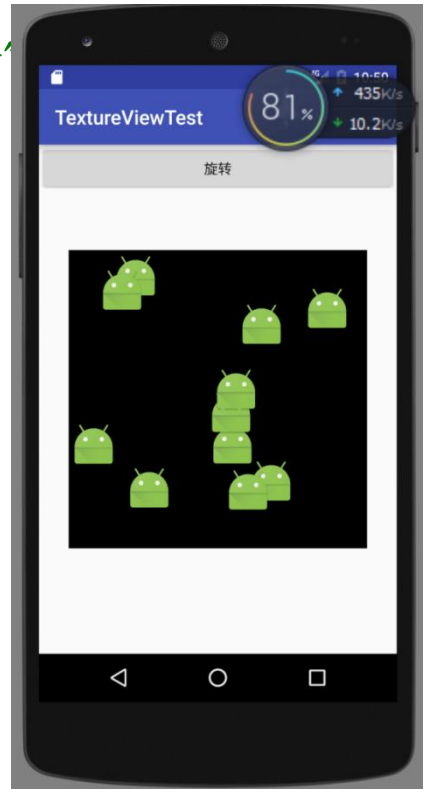
参考 参考

TextureView可以像View一样使用，也是在子线程绘制，但是只能使用在硬件加速开启的窗口中。

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.isszym.textureviewtest.MainActivity"
    tools:showIn="@layout/activity_main">

    <Button
        android:id="@+id/button_transform"
        android:text="旋转"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <TextureView
        android:id="@+id/surface"
        android:layout_gravity="center"
        android:layout_width="300dp"
        android:layout_height="300dp" />
</FrameLayout>
```



```
public class MainActivity extends AppCompatActivity implements View.OnClickListener,
        View.OnTouchListener, TextureView.SurfaceTextureListener{

    private TextureView mSurface;
    private DrawingThread mThread;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //  Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        //  setSupportActionBar(toolbar);

        findViewById(R.id.button_transform).setOnClickListener(this);
        mSurface = (TextureView)findViewById(R.id.surface);
        mSurface.setOnTouchListener(this);
        mSurface.setSurfaceTextureListener(this);
    }

    @Override
    public void onClick(View v) {
        // 旋转整个绘制视图
        mSurface.animate().rotation(mSurface.getRotation() < 180.f ? 180.f : 0.f);
    }
```

```
@Override
public boolean onTouch(View v, MotionEvent event) {
    if (event.getAction() == MotionEvent.ACTION_DOWN) {
        mThread.addItem((int)event.getX(), (int)event.getY());
    }
    return true;
}
@Override
public void onSurfaceTextureAvailable(SurfaceTexture surface, int width, int height){
    mThread = new DrawingThread(new Surface(surface),
            BitmapFactory.decodeResource(getResources(), R.mipmap.ic_launcher));
    mThread.updateSize(width, height);
    mThread.start();
}
@Override
public void onSurfaceTextureSizeChanged(SurfaceTexture surface,int width,int height){
    mThread.updateSize(width, height);
}
@Override
public boolean onSurfaceTextureDestroyed(SurfaceTexture surface) {
    mThread.quit();
    mThread = null;
    // 返回 true 并允许框架释放 Surface
    return true;
}
```

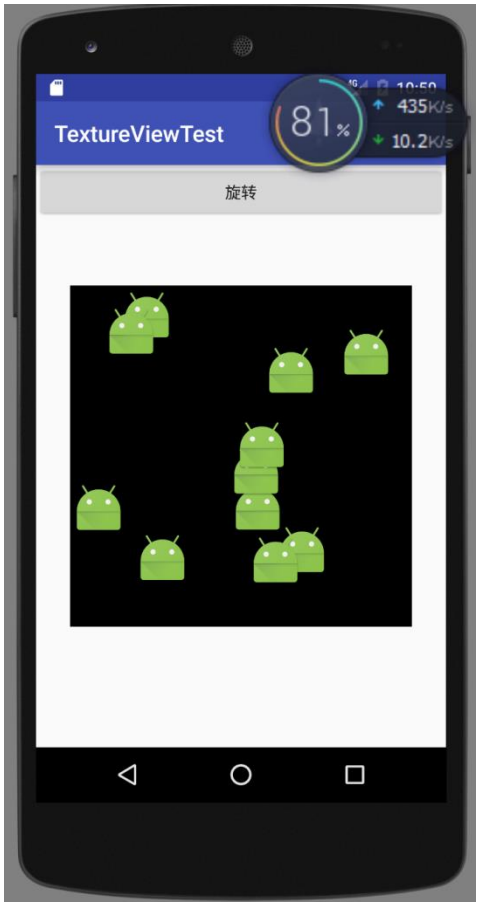
```
@Override
public void onSurfaceTextureUpdated(SurfaceTexture surface) { }
private class DrawingThread extends HandlerThread implements Handler.Callback{
    private static final int MSG_ADD = 100;
    private static final int MSG_MOVE = 101;
    private static final int MSG_CLEAR = 102;
    private int mDrawingWidth, mDrawingHeight;
    private boolean mRunning = false;
    private Surface mDrawingSurface;
    private Rect mSurfaceRect;
    private Paint mPaint;
    private Handler mReceiver;
    private Bitmap mIcon;
    private ArrayList<DrawingItem> mLocations;
    private class DrawingItem {
        int x, y; // 当前位置标识
        boolean horizontal, vertical; // 运动方向的标识

        public DrawingItem(int x, int y, boolean horizontal, boolean vertical) {
            this.x = x;
            this.y = y;
            this.horizontal = horizontal;
            this.vertical = vertical;
        }
    }
```

```
public DrawingThread(Surface surface, Bitmap icon) {
    super("DrawingThread");
    mDrawingSurface = surface;
    mSurfaceRect = new Rect();
    mLocations = new ArrayList<>();
    mPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
    mIcon = icon;
}

@Override
protected void onLooperPrepared() {
    mReceiver = new Handler(getLooper(), this);
    // 开始渲染
    mRunning = true;
    mReceiver.sendEmptyMessage(MSG_MOVE);
}

@Override
public boolean quit() {
    // 退出前清除所有的消息
    mRunning = false;
    mReceiver.removeCallbacksAndMessages(null);
    return super.quit();
}
```



```
@Override
public boolean handleMessage(Message msg) {
    switch (msg.what) {
        case MSG_ADD:
            // 在触摸的位置创建一个新的条目，该条目的开始方向是随机的
            DrawingItem newItem = new DrawingItem(msg.arg1, msg.arg2,
                    Math.round(Math.random()) == 0,
                    Math.round(Math.random()) == 0);
            mLocations.add(newItem);
            break;
        case MSG_CLEAR:
            // 删除所有的对象
            mLocations.clear();
            break;
        case MSG_MOVE:
            // 如果取消，则不做任何事情
            if (!mRunning) return true;

            // 渲染一帧
            try {
                // 锁定 SurfaceView，并返回到要绘图的 Canvas
                Canvas canvas = mDrawingSurface.lockCanvas(mSurfaceRect);
                // 首先清空 Canvas
                canvas.drawColor(Color.BLACK);
```

```
                for (DrawingItem item : mLocations) {// 绘制每个条目
                    // 更新位置
                    item.x += (item.horizontal ? 5 : -5);
                    if (item.x >= (mDrawingWidth - mIcon.getWidth())) {
                        item.horizontal = false;
                    }
                    if (item.x <= 0) {
                        item.horizontal = true;
                    }
                    item.y += (item.vertical ? 5 : -5);
                    if (item.y >= (mDrawingHeight - mIcon.getHeight())) {
                        item.vertical = false;
                    }
                    if (item.y <= 0) {
                        item.vertical = true;
                    }
                    canvas.drawBitmap(mIcon, item.x, item.y, mPaint);
                }
                // 解锁 Canvas，并渲染当前的图像
                mDrawingSurface.unlockCanvasAndPost(canvas);
            } catch (Exception e) {
                e.printStackTrace();
            }
            break;
    }
    if (mRunning) {// 发送下一帧
        mReceiver.sendEmptyMessage(MSG_MOVE);
    }
    return true;
}
```

```
    public void updateSize(int width, int height) {
            mDrawingWidth = width;
            mDrawingHeight = height;
            mSurfaceRect.set(0, 0, mDrawingWidth, mDrawingHeight);
        }

        public void addItem(int x, int y) {
            // 通过 Message 参数将位置传给处理程序
            Message msg = Message.obtain(mReceiver, MSG_ADD, x, y);
            mReceiver.sendMessage(msg);
        }

        public void clearItems() {
            mReceiver.sendEmptyMessage(MSG_CLEAR);
        }
    }
}
```