**Android 图形绘制和动画**

**Demo实例:**

****

* 1. **图形绘制**

1. **Bitmap**
2. 将Bitmap对象包装BitmapDrawable对象

BitmapDrawable \_bitmapDrawable = new BitmapDrawable(bitmap);

1. 从BitmapDrawable对象中获取Bitmap

Bitmap bitmap = \_bitmapDrawable.getBitmap();

1. 创建Bitmap对象
2. 从源位图source中获取从坐标(x, y)开始，像素宽width、高height的一块作为新的Bitmap

public static Bitmap createBitmap(Bitmap source, int x, int y, int width, int height)；

1. 从源位图source中获取从坐标(x, y)开始，像素宽width、高height的一块作为新的Bitmap，并按matrix指定的规则进行变换。

public static Bitmap createBitmap(Bitmap source, int x, int y, int width, int height, Matrix matrix, boolean filter)

1. 对源位图source进行缩放，缩放成像素宽dstWidth、高dstHeight作为新的Bitmap

public static Bitmap createScaledBitmap(Bitmap source, int dstWidth, int dstHeight, boolean filter)

1. Bitmap使用完之后，为了避免内存泄露引发OutOfMemory错误，要及时回收

public final boolean isRecycled()； //判读Bitmap是否已被回收

public void recycle()； //强制一个Bitmap对象立即回收自己

1. **BitmapFactory**
2. 从字节数组data的offset位置开始，获取长度为length的字节数据解析成Bitmap对象

public static Bitmap decodeByteArray(byte[] data, int offset, int length, Options opts) ;

1. 从pathName指定的文件中解析、创建Bitmap对象

public static Bitmap decodeFile(String pathName) ;

1. 从FileDescriptor对应的文件中解析、创建Bitmap对象

public static Bitmap decodeFileDescriptor(FileDescriptor fd, Rect outPadding, Options opts) ;

1. 根据指定的资源ID从指定资源中解析、创建Bitmap对象

public static Bitmap decodeResource(Resources res, int id);

1. 从指定的InputStream输入流中解析、创建Bitmap对象

public static Bitmap decodeResource(InputStream is);

1. **Canvas画布**
2. Android绘图应该继承View组件，并重写onDraw(Canvas canvas)方法
3. Canvas的绘图方法：

public final void rotate(float degrees, float px, float py)；//对Canvas旋转变换

public final void scale(float sx, float sy, float px, float py)；//对Canvas缩放变换

public native void skew(float sx, float sy); //对Canvas倾斜变换

public native void translate(float dx, float dy); //移动Canvas

****

1. **Paint画笔**

Paint类主要用于设置绘制风格，包括画笔颜色、画笔粗细、填充风格等。



PathEffect可以用于定义绘制效果，PathEffect的子类有：

CornerPathEffect 用圆角来代替尖锐的角，从而将尖锐的边角进行平滑。

DashPathEffect 创建一个虚线的轮廓(短横线/小圆点)，而不是使用实线

DiscretePathEffect 与DashPathEffect相似，但是添加了随机性。当绘制时，需要指定每一段的长度和与原始路径的偏离度

PathDashPathEffect 可以定义一个新的形状(路径)并将其用作原始路径的轮廓标记。

SumPathEffect 顺序地在一条路径中添加两种效果，，这样每一种效果都可以应用到原始路径中，而且两种结果可以结合起

ComposePathEffect将两种效果组合起来应用，先使用第一种效果，然后在这种效果的基础上应用第二种效果。

1. **图形特效处理**
2. Matrix控制变换
3. public void setTranslate(float dx, float dy); //控制Matrix进行平移。
4. public void setSkew(float kx, float ky); //Matrix沿x轴、y轴分别倾斜kx、ky。
5. public void setSkew(float kx, float ky, float px, float py); //Matrix以px、py为轴心，沿x轴、y轴分别倾斜kx、ky。
6. public void setRotate(float degrees, float px, float py)； //Matrix以px、py为轴心，旋转degrees角度
7. public void setScale(float sx, float sy, float px, float py)； //Matrix以px、py为轴心进行缩放，sx、sy分别为x轴、y轴方向的缩放比例。
8. Canvas通过调用drawBitmap(Bitmap bitmap, Matrix matrix, Paint paint)方法，可以在绘制bitmap时应用Matrix进行变换。
9. 扭曲图像Canvas中的drawBitmapMesh方法

public void drawBitmapMesh(Bitmap bitmap, int meshWidth, int meshHeight, float[] verts, int vertOffset, int[] colors, int colorOffset, Paint paint)；

bitmap：需要扭曲的源位图；

meshWidth：源位图在横向方向的等分的格子数；

meshHeight：源位图在纵向方向的等分的格子数；

verts：长度为2\*（meshWidth+1）\*（meshHeight +1）的数组，记录了各个顶点和网格线的交点位置；

vertOffset：控制verts数组从第几个数组元素开始对bitmap进行扭曲；

1. Shader填充图形

BitmapShader：使用位图平铺的渲染效果；

LinearGradient：使用线性渐变填充图形；

RadialGradient：使用圆形渐变填充图形；

SweepGradient：使用角度渐变填充图形；

ComposeShader：使用组合渲染效果来填充图形。

* 1. **动画**

1. **逐帧动画（Frame Animation**）

将静态图片按一帧一帧的播放，与电影播放原理完全一样

1. 在res/anim文件夹下创建my\_anim.xml文件

<?xml version="1.0" encoding="utf-8"?>

<animation-list xmlns:android=<http://schemas.android.com/apk/res/android>

android:oneshot="false">

<item android:drawable="@drawable/butterfly\_1" android:duration="60"/>

<item android:drawable="@drawable/butterfly\_2" android:duration="60"/>

</animation-list>

1. 将my\_anim设置成ImageView的背景图片
2. 调用AnimationDrawable的start()方法开始动画和stop()方法停止动画

注：android: oneshot控制是否循环播放，true不循环，false为循环播放。

1. **补间动画（Tween Animation）**

指定动画开始和动画结束的关键帧，动画的变化过程由系统计算、并补齐。

1. 透明度、大小、旋转度、位置改变的补间动画

AlphaAnimation：透明度改变动画，透明度变化范围从0到1；

ScaleAnimation：大小缩放动画；

RotateAnimation：旋转动画。

TranslateAnimation：位移变化动画；

TranslateAnimation \_translateAnimation =

new TranslateAnimation (fromXDelta, toXDelta, fromYDelta, toYDelta)；

\_translateAnimation.setDuration(ANIMA\_TIME); //设置动画时间

\_ translateAnimation.setFillAfter(true); //设置动画终止时停留在最后一帧

ImageView iv\_ image = (ImageView)findViewById(R.id.iv\_image);

//开始动画，动画结束imageView回到原来位置。要想结束时，imageView不回到初始位置，可以在AnimationListener监听器的onAnimationEnd()中使用Layout()方法调整imageView的位置。

iv\_ image.startAnimation(\_translateAnimation);

程序清单：res/anim/anim.xml

<?xml version="1.0" encoding="utf-8"?>

<set xmlns:android="http://schemas.android.com/apk/res/android"

android:interpolator="@android:anim/linear\_interpolator">

<scale android:fromXScale="1.0"

android:toXScale="0.01"

android:fromYScale="1.0"

android:toYScale="0.01"

android:pivotX="50%"

android:pivotY="50%"

android:fillAfter="true"

android:duration="3000"/>

<alpha android:fromAlpha="1"

android:toAlpha="0.05"

android:duration="3000"/>

<rotate android:fromDegrees="0"

android:toDegrees="1800"

android:pivotX="50%"

android:pivotY="50%"

android:duration="3000"/>

</set>

Animation anim =

AnimationUtils.loadAnimation(context, R.anim.anim); anim.setFillAfter(true);

ImageView iv\_ image = (ImageView)findViewById(R.id.iv\_image);

iv\_ image.startAnimation(anim); //开始动画

Interpolator根据特定算法计算出整个动画所需要动态插入帧的密度和位置。

Interpolator的实现类：

LinearInterpolator：动画以均匀速度改变；

AccelerateInterpolator：动画以加速度改变，开始较慢，然后加速；

AccelerateDecelerateInterpolator：动画开始和结束位置变化速度较慢，中间变化较快；

DecelerateInterpolator：动画以负加速度改变，开始较快，然后减速；

CycleInterpolator：动画循环播放特定次数，变化速度按正玄曲线变化。

1. 自定义补间动画

Animation作为补间动画的抽象类，提供了AlphaAnimation、RotateAnimation、ScaleAnimation、TranslateAnimation四个实现类。

自定义补间动画要继承Animation类，关键是要重写Animation的applyTransformation( float interplatedTime, Transformation transformation)方法。

interplatedTime：代表动画时间进行比；

transformation：补间动画在不同时间对图形或组件的变形程度。

为了控制图片或View进行三维空间的变换，需要借助Android的Camera类（并不是手机摄像头，只是一个空间变换工具），作用类似Matrix。

Camera类的方法（主要用于支持三维空间的变换）：

getMatrix(Matrix matrix)：将Camera所做的变换应用到Matrix上；

rotateX(float deg) /rotateY(float deg) /rotateZ(float deg)：将目标组件沿x、y、z轴旋转；

translate(float x, float y, float z)：将目标组件在三维空间进行位移变换；

applyToCanvas(Canvas canvas)：把Camera变换应用到Canvas。

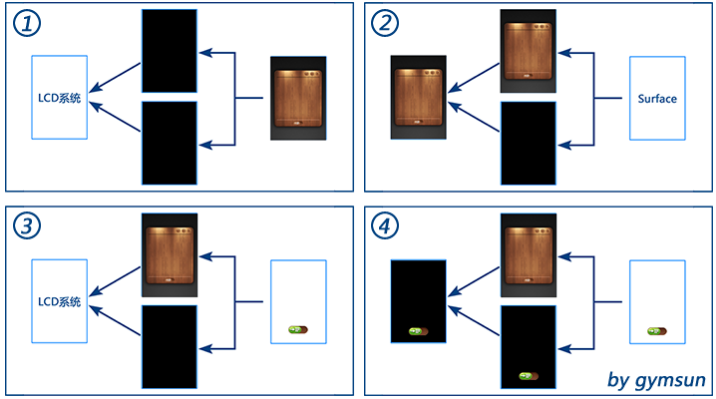
1. **属性动画**

属性动画可以看成增强版的补间动画。

1. 属性动画与补间动画类似，需要定义以下属性：
2. 动画持续时间，默认300ms，属性动画资源文件中通过android:duration属性指定。
3. 动画插值方式，与补间动画作用类似，在属性动画资源文件通过android:interpolator指定。
4. 动画重复次数，在属性动画资源文件通过android:repeatCount指定。
5. 动画重复行为，动画结束后、重复下次动画时，是再次从开始帧播放到结束帧还是从结束帧播放到开始帧，在属性动画资源文件通过android:repeatMode指定。
6. 动画集，将多个属性动画合成一组，既可让这组属性动画按次序播放，也可同时播放，通过android:ordering来指定。在属性动画资源文件通过<set …/>元素来组合动画集。
7. 帧刷新频率，指定帧播放频率，默认为10ms。
8. Animator是创建属性动画的基类，其三个实现类：
9. ValueAnimator：是Animator的子类，属性动画的时间引擎，负责计算各个帧的属性值。属性动画主要由两方面组成 A. 计算各帧相关属性； B. 为指定对象设置这些计算后的值。ValueAnimator只负责计算相干属性，程序员必须根据ValueAnimator计算并监听值更新来更新对象的相干属性。
10. ObjectAnimator：是ValueAnimator的子类，允许程序员对指定对象的属性执行动画。
11. AnimatorSet：是Animator的子类，用于组合对个Animator，并指定多个Animator是次序播放还是同时播放。
12. 属性动画Evaluator（计算器），控制属性动画如何计算属性值，有以下几类Evaluator：
13. IntEvaluator：用于计算int类型属性值；
14. FloatEvaluator：用于计算float类型属性值；
15. ArgbEvaluator：用于计算以十六进制形式表示的颜色值；
16. TypeEvaluator：计算器接口，开发者通过实现该接口实现自定义计算器。
17. 使用ValueAnimator创建动画的步骤：
18. 调用ValueAnimator静态方法ofInt()、ofFloat()或ofObject()获取ValueAnimator实例；
19. 调用ValueAnimator的setXxx()设置动画持续时间、插值方式、重复次数等；
20. 为ValueAnimator注册AnimatorUpdateListener监听器，在该监听器中监听ValueAnimator计算出来的值的变化，并将这些值应用于指定对象。
21. 调用ValueAnimator的start()方法开启动画；
22. ObjectAnimator继承了ValueAnimator，可以将在动画过程中ValueAnimator计算出来的值应用到指定对象的指定属性上，不需要注册AnimatorUpdateListener监听器。
23. ObjectAnimator创建动画需要注意以下几点：
24. 为该对象对应属性提供相应的setter方法；
25. 调用ObjectAnimator的ofInt()、ofFloat()或ofObject()时，如果只提供一个参数，则被认为是结束值，此时需要为该属性提供一个getter方法，getter方法返回值被作为开始值；
26. 如果对话对象为view，为了能显示动画效果，需要在onAnimationUpdate()事件监听器中调用View.invalidate()方法。
27. 定义属性动画的两种方式：
28. ValueAnimator（或ObjectAnimator）的静态方法ofInt()、ofFloat()或ofObject()创建；
29. 使用资源文件来定义动画。
30. **SurfaceView实现动画**
31. View的绘制机制存在以下缺陷：
32. View缺乏双缓冲机制；
33. 当程序更新View上的图像时，必须重绘View上的整张图片；
34. 在新线程中无法直接更新View组件。
35. Android提供了SurfaceView代替View，在游戏绘图方面有更好的性能。
36. SurfaceView一般会与SurfaceHolder结合使用，通过SurfaceView.getHolder()方法获取与其关联的SurfaceHolder，SurfaceHolder用于向关联的SurfaceView上绘图。
37. SurfaceHolder获取Canvas对象：
38. Canvas lockCanvas()：锁定整个SurfaceView对象，获取该SurfaceView上的Canvas；
39. Canvas lockCanvas(Rect rect)： 锁定SurfaceView上Rect划分的区域，获取其Canvas。
40. unlockCanvasAndPost(Canvas canvas)释放绘图、提交所绘制的图形。
41. SurfaceView的双缓冲机制

参考：<http://blog.csdn.net/gymsun/article/details/8516823> **（见后面流程图）**

1. 在画第1帧前，调用lockCanvas()， Surface锁住“缓冲区0”的全部区域，往画布上画的所有图片和文字，都被送到“缓冲区0”中。
2. 画完第1帧，调用 unlockCanvasAndPost()，则会将“缓冲区0”中所有的图像都显示到屏幕上。
3. 画第2帧前，调用lockCanvas(Rect dirty)，这时Surface锁住“缓冲区1”的部分区域，也就只能在这个区域中画上新的按钮图片。此时，“缓冲区1”中大部分区域都是没有图像的（黑色），除了绘画的按钮区域之外。
4. 画完第2帧，调用 unlockCanvasAndPost()，则会将“缓冲区1”中所有的图像显示到屏幕上。
5. 继续使用SurfaceView绘图，则重复步骤1)到步骤4）。



**注**：由于SurfaceView的双缓冲机制，绘图之前先清屏两次，即连续调用两次lockCanvas()、unlockCanvasAndPost()