


绘图(101题)

版本：2018/5/15-1(18:30)

 绘图-思维导图

- 绘图(101题)
 - 基础知识
 - View的绘制顺序
 - Canvas
 - 图层
 - 图层的保存(save...)
 - 图层标志
 - 绘制
 - 颜色绘制(drawColor...)
 - 图形绘制(drawCircle...)
 - Bitmap绘制(drawBitmap/drawBitmapMesh)
 - Picture绘制
 - 范围裁剪(clipXXX)
 - 几何变换
 - canvas
 - Matrix
 - camera 3D变换
 - Path
 - 添加图形(addCircle...)
 - 绘制线(moveto...)
 - setFillType
 - Paint
 - 初始化
 - 颜色处理
 - setColor/ARGB()
 - setShader
 - setColorFilter
 - setXfermode
 - 效果类
 - 抗锯齿、风格、线条宽度、线头、拐角
 - 色彩优化
 - 背景和前景
 - Path相关
 - 设置路线效果

- 获取Path
- 文本绘制
 - Canvas
 - Paint
 - 文本绘制辅助
 - 文字尺寸测量
 - 光标
- Bitmap
 - 颜色矩阵
 - 像素点
 - 像素块
- Matrix
 - Matrix原理
 - API
 - 构造方法和基本方法
 - 几何变换
 - 进阶方法(pre/post)
 - 其他方法
 - setRectToRect
 - setPolyToPoly
 - mapXXX
- 贝塞尔曲线
 - 基础与原理
 - 实际使用
- Region
 - 构造方法
 - 替换方法
 - RegionIterator
 - 区域操作
- Picture
- 参考资料

基础知识

1、手机的屏幕大小是什么(如4.7寸)?

1. 屏幕大小指的是屏幕对角线的长度，使用“寸”度量，4.7寸手机、5.5寸手机

2、PPI(DPI)是什么?

1. PPI：每英寸像素(Pixels Per Inch)又被称为DPI，对角线的像素点数除以屏幕的大小得到的。

3、dp是什么？

- 1. 独立像素密度dp
- 2. Android用mdpi即密度值为160的屏幕作为标准，在这个屏幕上1px = 1dp。例如100dp的长度，在mdpi中为100px，而在hdpi中为150px。
- 3. 因此**dp**用于屏幕适配最好。

4、dp和px换算表

比例换算表	mdpi	hdpi	xhdpi	xxhdpi
dp	1	1	1	1
px	1	1.5	2	3

View的绘制顺序

5、View的自定义绘制包含哪些部分

- 1. 方式：重写绘制方法（ onDraw ）
- 2. Canvas 的绘制类方法： drawXXX()-关键参数Paint
- 3. Canvas 的辅助类方法： clipXXX()-范围裁切 和 几何变换-Matrix
- 4. 使用 不同绘制方法 来控制 遮盖关系

6、View的绘制顺序图

7、super.onDraw()的绘制顺序

- 1. super.onDraw() 如果是直接继承自 View ， 顺序并不重要。
- 2. super.onDraw() 如果是继承自 已有控件 就需要根据情况，需要在已有画面之上的就需要在 onDraw 后面进行绘制

8、在onDraw()方法后的妙用(前景)

- 1. 可以在原基础上进行一些 点缀效果
- 2. 比如可以在 DEBUG 模式中，在 图片 上绘制尺寸效果

```
public class AppImageView extends ImageView {
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        if (DEBUG) {
            // 在 debug 模式下绘制出 drawable 的尺寸信息
            ...
        }
    }
}
```

9、在onDraw()方法后的使用(背景)

1. 主要是用于背景的绘制，使用场景较少一些。

10、dispatchDraw()

1. onDraw只是负责自身内容的绘制,如果有子View的绘制内容会导致 onDraw() 的效果可能被覆盖掉。
2. dispatchDraw() 会遍历子View并进行绘制。
3. 不想被子View 遮挡的内容，需要在 dispatchDraw() 之后调用

11、drawBackground()用于绘制背景，不能重写。

1. 通过 xml 的 background 属性绘制。
2. View.setBackground() 进行设置

12、onDrawForeground()-API23才加入

1. 滑动边缘渐变 和 滚动条
2. 前景绘制
3. onDrawForeground() 之后绘制的内容会 盖住 前景、滑动边缘渐变以及滑动条
4. onDrawForeground() 之前绘制的内容会处于 前景、滑动边缘渐变、滑动条 和 子View 之间

13、draw()方法的作用

1. draw() 是所有方法的总调度。
2. draw() 会依次执行 drawBackground()、onDraw()、dispatchDraw()、onDrawForeground()
3. Editext 想要更改背景色，就需要在 draw() 前更改背景色，不然可能会导致 下方的黑色线 消失。

14、ViewGroup子类会的注意事项

1. 在 ViewGroup 的子类中重写 除 dispatchDraw() 以外的绘制方法时，可能需要调用 setWillNotDraw(false);

Canvas

图层

15、图层是什么？

1. 类似于Photoshop图层概念。
2. 可以让 绘制 在图层中完成，而不会直接 绘制在屏幕上

16、为什么需要图层

1. 在处理 xfermode 的时候，原canvas 上面的图(包括背景)会影响到 src和dst 的合成。

图层的保存(save...)

17、图像的保存API

```
/**
 * 保存与恢复
 */
canvas.save(); //将当前的Matrix和Clip保存起来。在Restore之前的所有平移、缩放等操作
canvas.save(MATRIX_SAVE_FLAG|CLIP_SAVE_FLAG); //与上面的效果相同
/**
 * 恢复到上一图层的状态(Matrix/Clip):
 * 1. 将save与restore期间的绘制内容生效到上一图层
 * 2. 但期间改变的Matrix/Clip会丢弃。
 */
canvas.restore();
canvas.restoreToCount(saveCount); //恢复saveCount层图层(saveCount必须>=1)
/**
 * 保存图层并创建新图层(离屏):
 * =save(), 会重定向到offscreen Bitmap上, 后续的绘制都绘制在新的透明图层上。
 * restore()之后会绘制到上一个图层或者屏幕上。
 */
canvas.saveLayer(new RectF(), new Paint());
canvas.saveLayer(new RectF(), new Paint(), MATRIX_SAVE_FLAG); //第三个参数为Flags
canvas.saveLayer(0, 0, 500, 500, new Paint());
canvas.saveLayer(0, 0, 500, 500, new Paint(), MATRIX_SAVE_FLAG); //第三个参数为Flags
/**
 * 保存图层并创建新图层(具有透明度):
 * 后续绘制的内容都会带有透明度
 */
canvas.saveLayerAlpha(new RectF(), new Paint());
canvas.saveLayerAlpha(new RectF(), new Paint(), MATRIX_SAVE_FLAG); //第三个参数为Flags
canvas.saveLayerAlpha(0, 0, 500, 500, new Paint());
canvas.saveLayerAlpha(0, 0, 500, 500, new Paint(), MATRIX_SAVE_FLAG); //第三个参数为Flags
```

图层标志

18、图层标志Flags的作用

Flag	意义	适用方法
MATRIX_SAVE_FLAG	只保存图层的matrix矩阵	save, saveLayer
CLIP_SAVE_FLAG	只保存大小信息	save, saveLayer
ALL_SAVE_FLAG	保存所有信息	save, saveLayer

Flag	意义	适用方法
HAS_ALPHA_LAYER_SAVE_FLAG	给新图层添加透明通道，合并图层时，新图层区域内没有绘制的地方完全透明，会显示上一层的内容。	saveLayer
FULL_COLOR_LAYER_SAVE_FLAG	完全保留该图层颜色，合并图层时，新图层区域内的内容完全覆盖上一层(区域之外的还是显示上一层的内容)	saveLayer
CLIP_TO_LAYER_SAVE_FLAG	创建图层时，会把canvas（所有图层）裁剪到参数指定的范围，restore后依旧只有裁剪后的大小，能极大提高性能。	saveLayer

绘制

19、Canvas有三层对颜色的处理

-
- 1. [Canavs]基本颜色处理-Canvas的drawColor、drawBitmap以及用 paint(setColor和使用Shader着色器) 绘制颜色
 - 2. [Paint]ColorFilter-颜色过滤 Paint.setColorFilter(ColorFilter)
 - 3. [Paint]Xfermode- Paint.setXfermode(Xfermode)

颜色绘制(drawColor...)

20、drawColor/drawRGB/drawARGB/drawPaint

```
//纯色
canvas.drawColor(Color.CYAN);
//具有透明度
canvas.drawColor(Color.parseColor("#5500aa00"));
//通过RBG值设置颜色(范围：0~255)
canvas.drawRGB(255, 0, 0);
//通过ARGB值设置颜色
canvas.drawARGB(100, 0, 0, 255);

//给Canavs的Bitmap用特定画笔进行填充
canvas.drawPaint(paint);
```

图形绘制(drawCircle...)

21、圆形

```

//蓝色圆
Paint paint = new Paint();
paint.setColor(Color.BLUE);
canvas.drawCircle(200, 200, 180, paint);
//空心的线宽为10的圆
paint.reset();
paint.setStyle(Paint.Style.STROKE);
paint.setStrokeWidth(30); //单位为像素
canvas.drawCircle(600, 200, 180, paint);
//开启抗锯齿的圆（抗锯齿会导致图形的失真）
paint.reset();
paint.setStyle(Paint.Style.FILL);
paint.setAntiAlias(true);
canvas.drawCircle(200, 580, 180, paint);

```

22、矩形

```

//1. 绘制填充型矩形
Paint paint = new Paint();
paint.setStyle(Paint.Style.FILL);
canvas.drawRect(0, 0, 300, 200, paint);
//2. 通过Rect绘制矩形(Rect参数为整数)
paint.setStyle(Paint.Style.STROKE);
paint.setStrokeWidth(20);
Rect rect = new Rect(320, 0, 700, 150);
canvas.drawRect(rect, paint);
//3. 通过RectF绘制矩形
RectF rectF = new RectF(0, 255.4f, 288.88f, 466.66f);
paint.setColor(Color.parseColor("#88880000"));
paint.setStyle(Paint.Style.FILL_AND_STROKE);
canvas.drawRect(rectF, paint);

//4. 绘制圆角矩形
paint.setColor(Color.MAGENTA);
paint.setStyle(Paint.Style.FILL);
// API >= 21
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
    //第4\5参数分别为圆角在横向和纵向的半径
    canvas.drawRoundRect(320, 250, 650, 450, 20, 20, paint);
}else{
    rectF = new RectF(320, 250, 650, 450);
    //第2\3参数分别为圆角在横向和纵向的半径
    canvas.drawRoundRect(rectF, 20, 20, paint);
}

```

23、点

```

//1. BUII 方形原点
Paint paint = new Paint();
paint.setColor(Color.BLUE);
paint.setStrokeCap(Paint.Cap.BUTT);
paint.setStrokeWidth(50);
canvas.drawPoint(60, 60, paint);
//2. SQUARE 方点
paint.setColor(Color.RED);
paint.setStrokeCap(Paint.Cap.SQUARE);
canvas.drawPoint(200, 60, paint);
//3. ROUND 圆点
paint.setColor(Color.GRAY);
paint.setStrokeCap(Paint.Cap.ROUND);
canvas.drawPoint(60, 200, paint);
//4. 多个点-1(两个数组成一点)
paint.setColor(Color.CYAN);
paint.setStrokeWidth(30);
float[] points = {60, 400, 140, 400, 220, 400, 300, 400};
canvas.drawPoints(points, paint);
//5. 多个点-2(会去除前offset个float数值, 并按照count取接下来相应数量的float数值)
float[] points2 = {60, 500, 140, 500, 220, 500, 300, 500};
canvas.drawPoints(points2, 2, 4, paint);

```

24、椭圆

```

Paint paint = new Paint();
//1. 绘制椭圆
RectF rectF = new RectF(50, 50, 350, 200);
canvas.drawOval(rectF, paint);
//2. API >= 21 才可以用
paint.setColor(Color.BLUE);
canvas.drawOval(50, 300, 350, 550, paint);

```

25、线

```

Paint paint = new Paint();
paint.setStrokeWidth(20);
//1. 一条线
canvas.drawLine(10, 10, 300, 100, paint);
//2. 一组线
float[] lines = {10, 110, 300, 110,
                 100, 50, 100, 300};
paint.setColor(Color.RED);
canvas.drawLines(lines, paint);
//3. 一组线(去除offset个float数值, 再选取count个float数值作为线的数组)
float[] lines2 = {10, 210, 200, 210,
                  50, 150, 50, 300,
                  40, 300, 200, 340};
paint.setColor(Color.GREEN);
canvas.drawLines(lines2, 4, 8, paint); //没有画第一条线, 只有后面两条

```


26、弧线和扇形

```
Paint paint = new Paint();
//类似于椭圆等API >= 21时才可以用类似API(否则用RectF版本)
paint.setStyle(Paint.Style.FILL);
paint.setColor(Color.BLACK);
canvas.drawArc(50, 50, 400, 300, //所在椭圆的四个顶点
    10, //以X轴正方向为0度, 此处为10度
    130, //扇形划过的度数, 相对于X轴正方向 = 10度+130度=140度
    false, //是否连接到圆心(true=扇形; false=弧线)
    paint);
//2. 填充 + 连接到圆心
paint.setStyle(Paint.Style.FILL);
paint.setColor(Color.RED);
canvas.drawArc(450, 50, 800, 300, 10, 130, true, paint);
//3. 勾线 + 不连接到圆心
paint.setStyle(Paint.Style.STROKE);
paint.setColor(Color.BLUE);
paint.setStrokeWidth(20);
canvas.drawArc(50, 450, 400, 700, 10, 130, false, paint);
//4. 勾线 + 连接到圆心
paint.setStyle(Paint.Style.STROKE);
paint.setColor(Color.GREEN);
paint.setStrokeWidth(20);
canvas.drawArc(450, 450, 800, 700, 10, 130, true, paint);
```

27、自定义图形: Path

```
//1. 获取路径
Path path = new Path();
path.addArc(200, 200, 400, 400, -225, 225);
path.arcTo(400, 200, 600, 400, -180, 225, false);
path.lineTo(400, 542);
//2. 用Path绘制
Paint paint = new Paint();
canvas.drawPath(path, paint);
```

Bitmap绘制(drawBitmap/drawBitmapMesh)

28、Canvas中Bitmap的绘制

```

Paint paint = new Paint();
paint.setStyle(Paint.Style.FILL);
paint.setStrokeWidth(10);
//1. 0,0位置开始绘制
canvas.drawBitmap(mBitmap, 0, 0, paint);
//2. 在`src`的Rect`的区域中绘制(src为RectF版本只是参数为float而已)
Rect rect = new Rect(0, 50, 500, 400);
canvas.drawBitmap(mBitmap, null, //Bitmap需要被绘制的子集
    rect, //Bitmap会被缩放并平移至Rect(参数为int)指明的去榆中
    paint);
//3. Matrix对图片处理(平移, 缩放, 倾斜, 旋转等等)
Matrix matrix = new Matrix();
matrix.preRotate(15, mBitmap.getWidth() / 2, mBitmap.getHeight() / 2); // 设置旋转, 后面2个参数为
canvas.drawBitmap(mBitmap, matrix, paint);
//4. 将Bitmap用网格分隔-实现扭曲效果
float verts[] = {10, 20};
canvas.drawBitmapMesh(mBitmap, 5, 5, //纵向和横向的格数
    verts, //扭曲后图片各顶点坐标(meshWidth + 1)*(meshHeight + 1)*2 + vertOffset
    0, //从第几个顶点开始扭曲, 一般为0
    null, //数组-设置顶点颜色, 可以和Bitmap对应像素的颜色相加。一般为null
    0, //从第几个顶点开始转换颜色, 一般为0
    null);

```

29、顶点绘制

```

//顶点操作
canvas.drawVertices(xxx);

```

Picture绘制

1、Picture绘制的作用

1. Picture 能录制 Canvas 的操作
2. Picture 的使用, 需要 关闭硬件加速
3. 通过 Picture 进行绘制, 会比调用相关API要效率更高。

2、Canvas绘制Picture

```

canvas.drawPicture(picture);

```

Picture 是一种 矢量图

范围裁剪(clipXXX)

29、Canvas范围裁剪的分类

1. clipRect-以矩形Rect进行裁剪

2. clipPath-根据路线进行裁剪

3. clipRegion-根据Region指定的区域进行裁剪

30、Canvas范围裁剪的实例

```
//0、Rect能对Bitmap图形缩放
Paint paint = new Paint();
Rect rect = new Rect(100, 100, 400, 400);
Rect clipRect = new Rect(100, 150, 400, 350);

canvas.save();
//1、裁剪
canvas.clipRect(clipRect);
//2、裁剪(clipPath)
Path path = new Path();
path.moveTo(100, 100);
path.lineTo(400, 100);
path.lineTo(250, 400);
path.close();
canvas.clipPath(path);

canvas.drawBitmap(mBitmap, null, rect, paint);
canvas.restore();
```

几何变换

canvas

31、canvas的api进行几何变换(都是对坐标系的操作)

```
//canvas内部通过Matrix进行几何变换，进行了API封装，方便使用
canvas.translate(); //平移
canvas.rotate(); //旋转
canvas.scale(); //缩放
canvas.skew(); //错切

//获取/设置matrix
canvas.getMatrix();
canvas.setMatrix();
```

可以通过 canvas 的 save/restore 进行存储和恢复。

Matrix

32、Matrix的几何变换

1. 平移、旋转、缩放、错切都和Canvas方法差不多(但是有pre和post区别)
2. Matrix 的

```

Matrix matrix = new Matrix();
//1、平移
matrix.preTranslate(100, 100);
matrix.postTranslate(100, 100);
matrix.setTranslate(100, 100);
//2、旋转(顺时针旋转度数, 以及圆心(px,py))
matrix.setRotate(45, 200, 200);
//3、缩放
matrix.setScale(1.3f, 1.3f, //横向/纵向缩放倍数
                200, 200); //缩放的圆心
//4、错切(x和y的错切系数)
matrix.setSkew(0, 0.5f);

//5. 自定义变换(将图形中的像素点映射到目标坐标)
float[] pointsSrc = {left, top, right, top, left, bottom, right, bottom};
float[] pointsDst = {left - 10, top + 50, right + 120, top - 90, left + 20, bottom + 30, right

matrix.reset();
matrix.setPolyToPoly(pointsSrc, //原点的坐标(x, y)
                    0, //pointsSrc中的偏移
                    pointsDst, //目标点的坐标(x, y)
                    0, 4); //点数

canvas.save();
//5、应用Matrix效果(使用concat最好)
//      canvas.setMatrix(matrix); //1. 会抛弃Canvas当前的变换(但不同系统可能不一致)
canvas.concat(matrix); //2. 当前Canvas的变换矩阵 * Matrix = 两个变换都会生效
canvas.drawBitmap(mBitmap, null, rect, paint);
canvas.restore();

```

camera 3D变换

33、Camera 3D变换

```

canvas.save();
Camera camera = new Camera();
//1. 保存came的初始状态
camera.save();
//2. 旋转
camera.rotateX(60);
//3. Camera应用到Canvas上(应用前需要移动到中心)
canvas.translate(250, 250);
camera.applyToCanvas(canvas);
canvas.translate(-250, -250); //恢复到原来的位置
//5. 平移
camera.translate( 10, 100, 200);
/**=====
 * 6. 设置虚拟相机的位置
 * 1-相机默认位置(0, 0, -8英寸): 8 * 72 = 576像素
 * 2-绘制内容过大,会出现图像投影过大的模糊效果
 * 3-换算单位固定72像素,会导致像素越大的手机,模糊越明显
 *=====*/
camera.setLocation(0, 0, -10); //z在负数上越小,摄像机越远,图像越接近没有旋转的样子
//7. Camaera恢复
camera.restore();
canvas.drawBitmap(mBitmap, null, rect, paint);
canvas.restore();

```

Path

34、Path是什么？作用？

- 1. 封装了复合几何路径(由直线部分、二次曲线和三次曲线组成)
- 2. 能通过 Canvas 的 drawPath 进行绘制(由Paint决定是填充还是线)
- 3. 能用于 clipping 裁剪过程
- 4. 也可以在 path 上绘制 Text文本

添加图形(addCircle...)

35、Path添加图形

形如 addXXX()

```

Path path = new Path();
//1. 圆
path.addCircle(100, 100, 80, Path.Direction.CW); //CW:clockwise(顺时针) CCW: counter-cl
//2. 椭圆
path.addOval(0, 0, 200, 250, Path.Direction.CW);
//3. 矩形
path.addRect(0, 0, 200, 300, Path.Direction.CW);
path.addRoundRect(0, 0, 200, 300, 20, 20, Path.Direction.CW);
//4. 弧线
path.addArc(0, 0, 200, 300, 10, 160); //抬着笔过去画弧线
//5. 添加path
Path path2 = new Path();
path.addPath(path2);

```

绘制线(moveto...)

36、Path画线(直线或曲线)

```

Path path = new Path();
//0. 设置起点-不会绘制横线
path.moveTo(20, 50);
//1. 画线
path.lineTo(100, 100); //绝对坐标:当前位置画线到(100, 100)
path.rLineTo(100, 200); //相对坐标:画线到(当前X + 100, 当前Y + 200)

//2. 贝塞尔曲线(quadTo/rQuadTo=绝对/相对)
path.quadTo(300, 200, //控制点
            400, 300); //终点

//3. 三次贝塞尔曲线(相对坐标-rCubicTo)
path.cubicTo(200, 200, //控制点1
            400, 400, //控制点2
            250, 250); //终点

//4. 直接到目标椭圆区域绘制椭圆
path.arcTo(300, 300, 500, 600, -10, 150,
            false); //forceMoveTo=true: 不留下移动痕迹; =false: 会留下移动痕迹
/**=====
 *5. 封闭路径: 当前位置绘制到起点
 * 1. Paint.Style = FILL / FILL_AND_STROKE
 * 会自动封闭路径, 不需要close
 *=====*/
path.close(); //!=lineTo(起点坐标)
//绘制路径
Paint paint = new Paint();
paint.setStyle(Paint.Style.STROKE);
paint.setStrokeWidth(10);
canvas.drawPath(path, paint);

```

setFillType

37、Path的辅助类方法 path.setFillType(Path.FillType.EVEN_ODD); :

1. EVEN_ODD : 在平面内任意一点向任意方向射出一条射线, 该射线和整个图形相交的次数, 如果是 奇数 则该点 在图形内 ; 如果是 偶数 则该点 不在图形内 。
2. WINDING : 在平面内任意一点向任意方向射出一条射线, 该线和整个图形的线 (都是有向线) 相交的点数(顺时针和逆时针点数一一抵消)如果有剩余则表示 在图形内 , 否则 不在图形内 。
3. INVERSE_EVEN_ODD 和 EVEN_ODD 规则相反。
4. INVERSE_WINDING 和 WINDING 规则相反

```
Path path = new Path();
path.addCircle(100, 100, 80, Path.Direction.CW);
path.addCircle(222, 100, 80, Path.Direction.CW);
//1. EVEN_ODD是奇内偶外原则
path.setFillType(Path.FillType.EVEN_ODD);
Paint paint = new Paint();
paint.setStyle(Paint.Style.FILL);
paint.setStrokeWidth(10);
canvas.drawPath(path, paint);
```

Paint

38、Paint类的作用和方法

1. 翻译为 颜料 , 可以理解为 画笔 的作用。
2. Paint 的API大致分为四类: 1-初始化; 2-颜色; 3-效果; 4-文字相关

初始化

39、Paint的初始化类方法

```
Paint paint = new Paint();
//1. 重置所有属性为默认值。比new性能高。
paint.reset();
//2. 从oldPaint中将所有属性复制到当前Paint
paint.set(oldPaint);
//3. 批量设置flags(如抗锯齿、抖动)
paint.setFlags(Paint.ANTI_ALIAS_FLAG | Paint.DITHER_FLAG);
paint.getFlags(); //相反能获得Flags
```

颜色处理

40、Canvas有三层对颜色的处理

1. 基本颜色处理-Canvas的drawColor、drawBitmap以及用 paint(setColor和使用Shader着色器) 绘制颜色

2. ColorFilter-颜色过滤 Paint.setColorFilter(ColorFilter)

3. Xfermode- Paint.setXfermode(Xfermode)

setColor/ARGB()

41、直接设置颜色

```
Paint paint = new Paint();
paint.setTextSize(200);
//1. setColor设置颜色
paint.setColor(Color.parseColor("#009966"));
//2. ARGB设置颜色
paint.setARGB(100, 0, 0, 0);
canvas.drawText("Feather", 200, 200, paint);
```

setShader

42、 Shader(着色器) 指定- Shader类 具有5种子类

```
Paint paint = new Paint();
paint.setTextSize(200);
//线性渐变(CLAMP-钳子模式; MIRROR-镜面模式; REPEAT-重复模式)
LinearGradient linearGradient = new LinearGradient(
    50, 50, 500, 500, //两个端点坐标
    Color.BLUE, Color.CYAN, //两个端点颜色
    Shader.TileMode.CLAMP); //着色规则
paint.setShader(linearGradient);
//辐射渐变---小贴士绘制文本给定的坐标是基线
RadialGradient radialGradient = new RadialGradient(
    400, 150, 100, //圆心和半径
    Color.BLUE, Color.RED, //中心颜色和边缘颜色
    Shader.TileMode.MIRROR);

paint.setShader(radialGradient);
//扫描渐变
SweepGradient sweepGradient = new SweepGradient(400, 150, //圆心坐标
    Color.BLUE, Color.RED);
paint.setShader(sweepGradient);
//Bitmap着色器
BitmapShader bitmapShader = new BitmapShader(mBitmap, Shader.TileMode.MIRROR, Shader.TileMode.CLAMP);
paint.setShader(bitmapShader);
//混合着色器(相同类型shader会在硬件加速下无效)
ComposeShader composeShader = new ComposeShader(linearGradient, bitmapShader, PorterDuff.Mode.SRC_OVER);
paint.setShader(composeShader);
canvas.drawCircle(200, 200, 200, paint);
```

setColorFilter

43、 PorterDuffMode一共有17种

1. 分为 Alpha Compositing透明度组合 和 Blending混合
2. Alpha组合 一共12种，均是由 Porter和Duff(姓) 联合发表的
3. Blending 混合

44、Paint颜色过滤-setColorFilter

```
Paint paint = new Paint();
/**=====
 * 1、光照效果：目标像素颜色-最终红色=R * mul.R / 0xff + add.R(其他颜色一致)
 *    mul=0x00ffff会导致红色为0，add=0x003000中绿色的0x30会导致整体绿色加强
 *=====*/
LightingColorFilter lightingColorFilter = new LightingColorFilter(0x00ffff, 0x003000);
paint.setColorFilter(lightingColorFilter);
// 2、基于PorterDuff.Mode将Color与源图进行处理(和ComposeShader的区别在于只能指定Color作为源图)
PorterDuffColorFilter porterDuffColorFilter = new PorterDuffColorFilter(Color.parseColor("#ff00ff"),
    PorterDuff.Mode.DST_IN);
paint.setColorFilter(porterDuffColorFilter);
// 3、颜色矩阵进行颜色处理
ColorMatrix colorMatrix = new ColorMatrix();
colorMatrix.setSaturation(0.11f); //设置饱和度
ColorMatrixColorFilter colorMatrixColorFilter = new ColorMatrixColorFilter(colorMatrix);
paint.setColorFilter(colorMatrixColorFilter);
//test
canvas.drawBitmap(mBitmap, 100, 100, paint);
```

setXfermode

45、Paint的setXfermode(最后一层颜色处理)

1. 全写 transfer mode
2. 作用：将你要绘制的内容作为源图像，以View中已有的内容作为目标图像，选取一个 PorterDuff.Mode 作为处理方案。
3. setXfermode 有两个注意点：1要使用off-screen buffer；2要控制好透明区域；

Paint的setXfermode

```
//离屏缓冲(View的setLayerType会直接把整个View都绘制在离屏缓冲中)
int saved = canvas.saveLayer(null, null, Canvas.ALL_SAVE_FLAG);

canvas.drawBitmap(mBitmap, 0, 0, paint);
Xfermode xfermode = new PorterDuffXfermode(PorterDuff.Mode.SRC_OVER);
paint.setXfermode(xfermode);
canvas.drawRect(200, 0, 600, 600, paint);
paint.setXfermode(null);

//恢复
canvas.restoreToCount(saved);
```

效果类

抗锯齿、风格、线条宽度、线头、拐角

46、Paint的效果类API(抗锯齿、风格、线条宽度等)

```
//1、抗锯齿
Paint paint = new Paint(Paint.ANTI_ALIAS_FLAG); //构造时使用
paint.setAntiAlias(true); //设置
//2、风格
paint.setStyle(Paint.Style.FILL); //填充
paint.setStyle(Paint.Style.STROKE); //画线
paint.setStyle(Paint.Style.FILL_AND_STROKE); //填充+画线
//3、线条宽度
paint.setStrokeWidth(0); //宽度为1，不会随着几何变换而改变宽度-发际线模式
paint.setStrokeWidth(1); //宽度为1，会因为几何变换而放大
paint.setStrokeWidth(40); //宽度可以为40等具体数值
//4、线头形状(像素=1时，三种线头完全一致；像素>1时，会产生不同)
paint.setStrokeCap(Paint.Cap.BUTT); //平头
paint.setStrokeCap(Paint.Cap.SQUARE); //平头(在BUTT的平头上要超出基准虚线一部分)
paint.setStrokeCap(Paint.Cap.ROUND); //圆头
//5、拐角形状
paint.setStrokeJoin(Paint.Join.MITER); //尖角(角长度受到控制)
paint.setStrokeJoin(Paint.Join.BEVEL); //平角
paint.setStrokeJoin(Paint.Join.ROUND); //圆角
//6、控制拐角中央角的角长度，超过一定长度就按照BEVEL平角处理(https://ws3.sinaimg.cn/large/
paint.setStrokeMiter(5); //(miter默认值 = 4，大约是29度锐角)
```

 paint_setStrokeMiter

色彩优化

47、Paint色彩优化

1. setDither-设置图像的抖动
2. setFilterBitmap-双线性过滤
3. 抖动的原理就是 图像 从 较高色彩深度 向 较低色彩深度 绘制时，在 图像 中加入 噪点 是的推向更真实(如：只有黑白两色的系统，通过黑格和白格交错，密度极大时产生灰色效果)

```
//开启抖动(优化色彩深度降低时的绘制效果)
paint.setDither(true);
//采用双线性过滤来绘制Bitmap(适合用于放大绘制的时候)
paint.setFilterBitmap(true);
```

背景和前景

48、Paint的设置背景阴影/设置前景效果

1-setShadowLayer-设置背景阴影

2-setMaskFilter-设置前景效果(需要关闭硬件加速)

```
/**=====
 * 给绘制内容加一层阴影(clearShadowLayer能清除阴影)
 * 1. 硬件加速的情况下，只支持文字绘制
 * 2. 如果参数中颜色有透明度，阴影的透明度就使用该透明度
 * 否则，阴影的透明度跟随Paint的透明度
 *=====*/
paint.setShadowLayer(20, //阴影的模糊范围
    10, 10, //阴影在xy上的偏移量
    Color.BLUE);
canvas.drawPath(path, paint);

/**=====
 * 1、模糊前景
 * 1-Normal: 内外都模糊
 * 2-Solid: 内外正常绘制，外部模糊
 * 3-Iner: 内部模糊，外部不绘制
 * 4-Outer: 内部不绘制，外部模糊
 *=====*/
paint.reset();
paint.setMaskFilter(new BlurMaskFilter(30, //模糊半径
    BlurMaskFilter.Blur.NORMAL));

/**=====
 * 2、浮雕效果
 *=====*/
paint.setMaskFilter(new EmbossMaskFilter(new float[]{0, 1, 1}, //x,y,z三个方向上的值，用
    0.2f, //环境光的强度，0~1
    8, //镜面反射系数，越接近0，反射光越强
    3 //模糊半径，值越大，越明显
));
canvas.drawBitmap(mBitmap, 100, 100, paint);
```

Path相关

设置路线效果

49、Paint设置路线效果(setPathEffect)

有七种：1、CornerPathEffect; 2、DsicretePathEffect; 3、DashPathEffect; 4、PathDashPathEffect 5、SumPathEffect 6、ComposePathEffect

- canvas.drawLine/s时，不支持硬件加速。

```

Paint paint = new Paint(); //构造时使用
paint.setStyle(Paint.Style.STROKE);
paint.setStrokeWidth(5);
Path path = new Path();
path.moveTo(100, 100);
path.rLineTo(300, 300);
path.rLineTo(300, -200);
path.rLineTo(300, 300);
//1、圆角路径
CornerPathEffect cornerPathEffect = new CornerPathEffect(20); //半径
paint.setPathEffect(cornerPathEffect);
//2、随机偏离
DiscretePathEffect discretePathEffect = new DiscretePathEffect(
    20, //每个小线段的长度
    5); //偏移值-越大越乱
paint.setPathEffect(discretePathEffect);
//3、虚线
DashPathEffect dashPathEffect = new DashPathEffect(
    new float[]{20, 10, 5, 10, 40, 10}, //必须为偶数，奇数位的值表示画几个像素，偶数位
    200); //整个虚线的偏移值
paint.setPathEffect(dashPathEffect);
//4、Path绘制虚线-需要关闭硬件加速
Path dashPath = new Path();
dashPath.rLineTo(30, 30);
dashPath.rLineTo(30, -30);
dashPath.close();

PathDashPathEffect pathDashPathEffect = new PathDashPathEffect(dashPath,
    30, //两个虚线path的起点的间隔
    50, //整个虚线的偏移值
    PathDashPathEffect.Style.TRANSLATE); //TRANSLATE:位移 ROTATE:旋转 MORPH:变体
paint.setPathEffect(pathDashPathEffect);
//5、组合效果-按两种PathEffect分别绘制
SumPathEffect sumPathEffect = new SumPathEffect(discretePathEffect, dashPathEffect);
paint.setPathEffect(sumPathEffect);
//6、组合效果-按照PathEffect依次绘制
ComposePathEffect composePathEffect = new ComposePathEffect(
    dashPathEffect, //后应用
    discretePathEffect); //先应用
paint.setPathEffect(composePathEffect);

canvas.drawPath(path, paint);

```

获取Path

50、Paint获取绘制的Path?

1. 绘制的Path 与初始的 Path 并不同，会计算上后期的各种处理。
2. 文字的Path 就是整个文字的边框(文字在绘制中转换为线进行绘制)
3. 主要用于 图形和文字 的装饰效果的 位置计算，如下划线

```
//1-获取到实际Path  
paint.setFillPath(srcPath, dstPath);  
//2-获取到文本Path  
paint.getTextPath("Hello", 0, 4, //字符串中字符范围  
    100, 100, //文本的起始坐标  
    dstPath);
```

文本绘制

Canvas

51、Canvas绘制文字的方式(3种)

```

/**=====
 * 1-文本的绘制
 * 1. 文本绘制以基线为准
 *=====*/
paint.setTextSize(50);
canvas.drawText("Hello Feather", 0, 100, paint);
// 2. 设置上下文和文字方向(中文和英文无影响)
canvas.drawTextRun("4 , 1 , عربي", //文本的start~end个字符
    1, 4, //上下文-contextstart <= start,contextend >= end
    0, 200, //文字坐标
    true, //文字从右到左
    paint);
// 3. 在路径上绘图
String str = "Harry Potter is a series of fantasy novels written by British author J.K.
paint.setStyle(Paint.Style.FILL);
canvas.drawTextOnPath(str,
    path,
    0, //文字水平偏移
    0, //文字垂直偏移
    paint);
// 4. 换行
TextPaint textPaint = new TextPaint();
textPaint.setTextSize(50);
String text = "Harry Potter is a series of fantasy novels written by British author J.K.
StaticLayout staticLayout = new StaticLayout(text, textPaint, //文本画笔
    600, //文字区域宽度, 达到会自动换行
    Layout.Alignment.ALIGN_NORMAL, //文字对齐方向
    1, //行间距倍数
    0, //行间距的额外增加值
    true); //文字上下添加额外的空间, 避免过高字符
String text2 = "Harry Potter \nis a series of fantasy novels \nwritten by British autho
StaticLayout staticLayout2 = new StaticLayout(text2, textPaint, 600, Layout.Alignment.AL

canvas.save();
canvas.translate(0, 100);
//达到宽度才换行
staticLayout.draw(canvas);
canvas.translate(0, 200);
//达到宽度和遇到“\n”都换行
staticLayout2.draw(canvas);
canvas.restore();

```

Paint

文本绘制辅助

52、Paint对文本绘制的辅助作用

```
//1. 字体大小
paint.setTextSize(60);
//2. 字体
paint.setTypeface(Typeface.SERIF);//内置
paint.setTypeface(Typeface.createFromAsset(getContext().getAssets(), "a.ttf")); //下载的
//3. 是否使用伪粗体
paint.setFakeBoldText(true);
//4. 是否添加删除线
paint.setStrikeThruText(true);
//5. 下划线
paint.setUnderlineText(true);
//6. 文字横向错切角度(倾斜)
paint.setTextSkewX(-0.3f);
//7. 文字横向缩放
paint.setTextScaleX(1.2f);
//8. 字符间距
paint.setLetterSpacing(0.05f);
//9. CSS的字体特性设置
paint.setFontFeatureSettings("smcp");
//10. 文字对齐方式
paint.setTextAlign(Paint.Align.CENTER); //左右中
//11. Local地域设置(不用在系统里设置)
paint.setTextLocale(Locale.CHINA);

canvas.drawText("Hello Feather", 0, 100, paint);
```

文字尺寸测量

53、Paint测量文字尺寸类

```
String text = "Hello Feather";
//1. 获取到推荐行距。方便换行。
canvas.drawText(text, 0, 100 + paint.getFontSpacing(), paint);
//2. top/bottom 任何文字的上下范围 ascent/descent 限制普通字符的顶部和底部
// top和ascent在baseline的上方，为负数；bottom和descent为正数
Paint.FontMetrics metrics = paint.getFontMetrics(); //leading就是上者bottom和下者top的距
paint.getFontMetrics(metrics); //不需要重新创建，性能更好
paint.ascent(); //能直接获取
paint.descent();
//3. 会之后获取文字的显示范围(第1~第3个字符)-存储在Rect中
Rect bound = new Rect();
paint.getTextBounds(text, 0, 2, bound); //不包含预留空隙
//4. 文本的宽度(包含左右看不见的预留空隙)
float width = paint.measureText(text);
//5. 每个字符的宽度
float[] width = new float[10];
paint.getTextWidths(text, width);
//6. 测量宽度，不够用就截断(可用于多行文本的折行计算)
float[] measuredWidth = {0};
int measuredCount = paint.breakText(text, 0, text.length(), true, 300, measuredWidth);
// 宽度上限 300 (不够用，截断)
canvas.drawText(text, 0, measuredCount, 150, 150, paint);
```

光标

54、Paint光标相关方法

```
String text = "Hello HenCoder \uD83C\uDDE8\uD83C\uDDF3";
//1. 计算光标的位置(具有emoji表情时不会出现在emoji中间)
float advance = paint.getRunAdvance(text, 0, text.length(), 0, text.length(), false, text);
canvas.drawText(text, 150, 150, paint);
canvas.drawLine(150 + advance, 150 - 50, 150 + advance, 150 + 10, paint);
//2. 获取到离某个位置最近的字符的offset(字符串中)
int offest = paint.getOffsetForAdvance(text, 0, text.length(), 0, text.length(), true, 100); //位置的像素值。该方法配合getRunAdvance能实现获取用户点击处的文字坐标的需求
//3. 判断一个字符串中是否是一个单独的字形glyph
paint.hasGlyph(text); //"a"=true;"b"=true;"ab"=false;"\uD83C\uDDE8\uD83C\uDDF3"=true;
```

Bitmap

55、Bitmap的简介

- 1、Bitmap包含了点阵和颜色值。
- 2、点阵就是包含像素的矩阵。
- 3、颜色值就是ARGB-对应透明度、红色、绿色、蓝色。

颜色矩阵

56、图像描述的三个角度：

- 1. 色调-物体显示的颜色
- 2. 饱和度-颜色的纯度。0(灰度)到100%(饱和)来进行描述。
- 3. 亮度-颜色相对的明暗程度。

57、ColorMatrix

- 1. Android使用的颜色矩阵。
- 2. 4X5的矩阵。

初始矩阵：不会对颜色产生任何变化

-	-	-	-	-	偏移量
决定R	1	0	0	0	0
决定G	0	1	0	0	0
决定B	0	0	1	0	0
透明度	0	0	0	1	0

58、改变颜色的三种方法

- 1-改变偏移量(红色和绿色分量都增加100，红绿结合，导致颜色偏黄)

1	0	0	0	100
0	1	0	0	100
0	0	1	0	0
0	0	0	1	0

- 2-改变颜色系数

1	0	0	0	0
0	2	0	0	0
0	0	1	0	0
0	0	0	1	0

- 3-改变色光属性

- 1. 色调、饱和度、亮度

2. ColorMatrix 封装了一些API来快速调整这些参数，不用计算矩阵值。

59、ColorMatrix的主要方法

```
/*-----
 * 处理色调：setRotate中0,1,2代表R,G,B
 * -----*/
ColorMatrix hueColorMatrix = new ColorMatrix();
hueColorMatrix.setRotate(0, hue);
hueColorMatrix.setRotate(1, hue);
hueColorMatrix.setRotate(2, hue);

/*-----
 * 设置饱和度
 * -----*/
ColorMatrix saColorMatrix = new ColorMatrix();
saColorMatrix.setSaturation(saturation);

/*-----
 * 设置亮度：将三原色相同比例混合显示出白色，以此提高亮度
 * -----*/
ColorMatrix lumColorMatrix = new ColorMatrix();
lumColorMatrix.setScale(lum, lum, lum, 1);

/*-----
 * 将矩阵作用效果混合，叠加效果。
 * -----*/
ColorMatrix colorMatrix = new ColorMatrix();
colorMatrix.postConcat(hueColorMatrix);
colorMatrix.postConcat(saColorMatrix);
colorMatrix.postConcat(lumColorMatrix);

/*-----
 * 设置矩阵，进行绘制
 * -----*/
paint.setColorFilter(new ColorMatrixColorFilter(colorMatrix));
canvas.drawBitmap(bitmap, 0, 0, paint);
```

60、ColorMatrix直接设置矩阵

```
private float[] mColorMatrix = new float[20];
xxxx
Bitmap bmp = Bitmap.createBitmap(bitmap.getWidth(), bitmap.getHeight(),
    Bitmap.Config.ARGB_8888);
Canvas canvas = new Canvas(bmp);
Paint paint = new Paint();
// 1. 设置矩阵
ColorMatrix colorMatrix = new ColorMatrix();
colorMatrix.set(mColorMatrix);
// 2. 进行绘制
paint.setColorFilter(new ColorMatrixColorFilter(colorMatrix));
canvas.drawBitmap(bitmap, 0, 0, paint);
mImageView.setImageBitmap(bmp);
```

像素点

61、像素点处理(Bitmap.getPixels/bitmap.setPixels)

1. 可以通过改变每个像素点的ARGB值，进行色彩处理。
2. Bitmap.getPixels(colorArray, 省略...) 可以提取整个图片的像素点，并保存到数组中。
3. 也是通过 颜色矩阵进行处理

62、像素点处理实例

```
r = Color.red(colorArray);
g = Color.green(colorArray);
b = Color.blue(colorArray);
a = Color.alpha(colorArray);
```

获取到原ARGB后，通过计算得到新的ARGB： $r1 = i * r + j * g + k * b$;

```
newPix[i] = Color.argb(a, r1, g1, b1);
bitmap.setPixels(newPix, ...);
```

使用不同参数，就能达到老照片、浮雕等效果。

像素块

63、canvas.drawBitmapMesh-利用像素块进行图形特效处理

1. 将图片分为像素块，改变像素块来处理图片。
2. canvas.drawBitmapMesh(bitmap, WIDTH, HEIGHT, verts, 0, null, 0, null); ,将第五个参数的坐标数组传入，bitmap会根据像素块数组，将WIDTH X HEIGHT网格中，每个焦点的坐标进行相应的修改。

```

public class FlagImage extends View {
    Bitmap bitmap;
    //定义两个常量,这两个常量指定该图片横向20格,纵向上都被划分为10格
    private final int WIDTH = 30;
    private final int HEIGHT = 30;
    //记录该图像上包含的231个顶点
    private final int COUNT = (WIDTH + 1) * (HEIGHT + 1);
    //定义一个数组,记录Bitmap上的21*11个点的坐标
    private final float[] verts = new float[COUNT * 2];
    //定义一个数组,记录Bitmap上的21*11个点经过扭曲后的坐标
    //对图片扭曲的关键就是修改该数组里元素的值
    private final float[] orig = new float[COUNT * 2];

    //振幅大小
    private final float A = 10;
    private float k = 0; //旗帜飞扬

    public FlagImage(Context context, @Nullable AttributeSet attrs) {
        super(context, attrs);
        init();
    }

    private void init() {
        bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.jide);

        float bitmapWidth = bitmap.getWidth();
        float bitmapHeight = bitmap.getHeight();
        int index = 0;
        for(int y = 0; y <= HEIGHT; y++){
            float fy = bitmapHeight * y / HEIGHT; //图像高度 乘以 y/总格数 可以获得 纵向y的坐标
            for(int x = 0; x <= WIDTH; x++){
                float fx = bitmapWidth * x / WIDTH; //x的坐标值

                orig [index * 2 + 0] = verts [index * 2 + 0] = fx; //[(x1,y1), (x2,y2), (...)..
                orig [index * 2 + 1] = verts [index * 2 + 1] = fy+100; //这里人为将坐标+100是为
                index += 1;
            }
        }
    }

    @Override
    protected void onDraw(Canvas canvas) {
        flagWave(); //旗帜飞扬
        k += 0.1F;
        //对bitmap按verts数组进行扭曲
        //从第一个点(由第5个参数0控制)开始扭曲
        canvas.drawBitmapMesh(bitmap, WIDTH, HEIGHT, verts, 0, null, 0, null);
        invalidate();
    }

    private void flagWave(){
        for(int j = 0; j <= HEIGHT; j++){
            for (int i = 0; i <= WIDTH; i++){
                verts[(j * (WIDTH + 1) + i) * 2 + 0] += 0; //x值不变
            }
        }
    }
}

```

```

        float offsetY = (float)Math.sin((float)i/WIDTH*2*Math.PI + Math.PI * k);
        verts[(j * (WIDTH + 1) + i) * 2 + 1] = orig[(j*WIDTH+i)*2+1]+offsetY*A;
    }
}

private void flagWave(float cx, float cy){
    for(int i = 0; i < COUNT * 2; i += 2)
    {
        float dx = cx - orig[i + 0];
        float dy = cy - orig[i + 1];
        float dd = dx * dx + dy * dy;
        //计算每个坐标点与当前点(cx,cy)之间的距离
        float d = (float)Math.sqrt(dd);
        //计算扭曲度, 距离当前点(cx,cy)越远, 扭曲度越小
        float pull = 80000 / ((float)(dd * d));
        //对verts数组(保存bitmap 上21 * 21个点经过扭曲后的坐标)重新赋值
        if(pull >= 1)
        {
            verts[i + 0] = cx;
            verts[i + 1] = cy;
        }
        else
        {
            //控制各顶点向触摸事件发生点偏移
            verts[i + 0] = orig[i + 0] + dx * pull;
            verts[i + 1] = orig[i + 1] + dx * pull;
        }
    }
    //通知View组件重绘
    invalidate();
}

public boolean onTouchEvent(MotionEvent event)
{
    // 调用warp方法根据触摸屏事件的坐标点来扭曲verts数组
    // flagWave(event.getX() , event.getY());
    return true;
}
}

```

Matrix

Matrix原理

64、Matrix是什么？与ColorMatrix的区别？

1. Matrix 是一个 3x3矩阵，用于 处理图形。
2. Matrix 可以进行平移变换、缩放变换、旋转变换、错切变换。
3. ColorMatrix 是一个 4x5矩阵，用于 处理颜色

65、Matrix矩阵的组成

$$\begin{pmatrix} MS_{SCALE_X} & MS_{KEW_X} & MTRANS_X \\ MS_{KEW_Y} & MS_{CALE_Y} & MTRANS_Y \\ MPERSP_0 & MPERSP_1 & MPERSP_2 \end{pmatrix}$$

1. scale-缩放
2. skew-错切
3. trans-平移
4. persp-表示透视

66、矩阵如何作用于图像

1. 对于坐标(x,y)用矩阵表述，z轴用1代表默认值(值越大就越远)

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

2. (x0,y0)旋转变换到(x,y)的转换过程:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \\ 1 \end{pmatrix}$$

67、什么是仿射矩阵

1. 仿射变换 其实就是二维坐标到二维坐标的线性变换，保持二维图形的“平直性”（即变换后直线还是直线不会打弯，圆弧还是圆弧）和“平行性”（指保持二维图形间的相对位置关系不变，平行线还是平行线，而直线上点的位置顺序不变）。
2. 仿射变换 可以通过一系列的原子变换的复合来实现，原子变换就包括： 平移、缩放、翻转、旋转和错切 。这里除了透视可以改变z轴以外，其他的变换基本都是上述的原子变换，
3. 所以，只要最后一行是0,0,1则是仿射矩阵。

API

构造方法和基本方法

68、Matrix的构造方法和基本方法

```

/**=====
 * 1、构造方法
 *=====*/
Matrix srcMatrix = new Matrix(); //会创建一个单位矩阵，3X3矩阵-左上角到右下角三个值均为1，其余为0
Matrix matrix = new Matrix(srcMatrix);
/**=====
 * 2、基本功能API
 *=====*/
matrix.isIdentity(); //是否是单位矩阵
matrix.isAffine(); //是否是仿射矩阵
matrix.rectStaysRect(); //是否可以将矩阵变换为矩形(单位矩阵，只进行平移、缩放、旋转90度倍数时，返回
matrix.reset(); //重置为单位矩阵
matrix.invert(srcMatrix); //获得目标矩阵的反转矩阵。srcMatrix*invert=单位矩阵

```

几何变换

69、Matrix的基本几何变换

```

matrix.setTranslate(); //平移
matrix.setScale(); //缩放
matrix.setSkew(); //错切
matrix.setRotate(); //旋转

```

70、Matrix效果叠加

```

matrix.setConcat(a, b);

```

将当前matrix的值为Matrix a 和 Matrix b 的乘积。(axb)

71、特殊的旋转变换setSinCos

```

setSinCos(float sinValue, float cosValue, float px, float py)
setSinCos(float sinValue, float cosValue)

```

sinValue: 对应图中的sin值

cosValue: 对应cos值

px:中心的x坐标

py: 中心的y坐标

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \\ 1 \end{pmatrix}$$

进阶方法(pre/post)

72、矩阵的前乘和后乘

1. Matrix 可以通过 前乘/后乘 让多个效果同时生效。

```
Matrix matrix = new Matrix();  
//2. 再进行平移  
matrix.setTranslate(100, 1000);  
//1. 前乘：先进行缩放  
matrix.preScale(0.5f, 0.5f);
```

如果是 后乘缩放 会导致 平移的距离也被缩放

73、Matrix的前乘和后乘API

```
/**=====
```

* 前乘

```
=====*/  
matrix.preTranslate(); //平移  
matrix.preScale();     //缩放  
matrix.preSkew();      //错切  
matrix.preRotate();    //旋转  
/**=====
```

* 后乘

```
=====*/  
matrix.postTranslate(); //平移  
matrix.postScale();     //缩放  
matrix.postSkew();      //错切  
matrix.postRotate();    //旋转
```

74、Matrix的几何变换-实例

使用矩阵Matrix对图片进行平移、缩放、旋转、错切变换。


```

public class PictureEditActivity extends Activity {

    ImageView imageView;
    static float data = 10f; //自定义的数据
    Bitmap bitmap;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_picture_edit);

        imageView = (ImageView) findViewById(R.id.pic_edit_imageview);
        bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.jide);
        //imageView.setImageBitmap(bitmap);

        Handler handler = new Handler();
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {
                editPic();
            }
        }, 1000);
    }

    public void editPic(){
        Canvas canvas = new Canvas();
        /*-----
        * 使用矩阵处理图片
        * -----*/
        float[] mImageMatrix = new float[9];
        Matrix matrix = new Matrix();
        matrix.setValues(mImageMatrix); //转为矩阵
        canvas.drawBitmap(bitmap, matrix, null);

        //set会覆盖整个矩阵
        matrix.setRotate(data); //旋转
        matrix.setTranslate(data, data); //平移
        matrix.setScale(data, data); // 缩放
        matrix.setSkew(data, data); //错切
        /*-----
        * 矩阵混合
        * pre()前乘: 当前矩阵 X 该次使用的矩阵
        * post()后乘: 该次使用的矩阵 X当前矩阵
        * -----*/
        matrix.preRotate(data); //会用 当前矩阵 X 该次的旋转矩阵
        matrix.postScale(data, data); // 本次缩放矩阵 X 当前矩阵
        /*
        * 1.先旋转45度
        * 2.再平移(200, 200)
        * */
        //后乘: 旋转后平移
        matrix.setRotate(45f);
        matrix.postTranslate(200, 200);
        //先乘: 平移前旋转
    }
}

```

```

matrix.setTranslate(200, 200);
matrix.preRotate(45f);

/*-----
 *      测试:
 * -----*/
Bitmap bmp = Bitmap.createBitmap(bitmap.getWidth(), bitmap.getHeight(),
    Bitmap.Config.ARGB_8888); // 创造bitmap的复制品bmp
canvas = new Canvas(bmp); // 用bmp创造画布
canvas.drawBitmap(bitmap, matrix, null); // 根据bitmap以matrix变化后的图片, 在bmp上进行绘制
imageView.setImageBitmap(bmp);
    }
}

```

其他方法

setRectToRect

75、setRectToRect的作用

```
matrix.setRectToRect(RectF src, RectF dst, ScaleToFit stf);
```

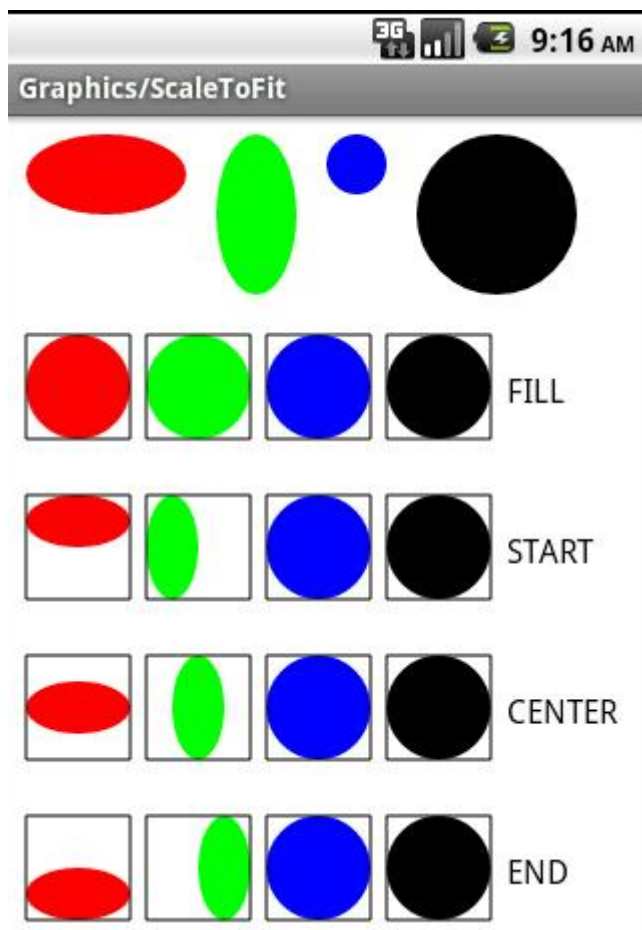
将当前 Matrix 设置为能够将 src 根据 ScaleToFit 缩放平移到 dst 所需要的 矩阵

76、ScaleToFit的取值

```

Matrix.ScaleToFit.CENTER; //保持坐标变换前矩形的长宽比, 并最大限度的填充变换后的矩形。至少保持一边
Matrix.ScaleToFit.FILL; //可能会改变长宽比, 最终高度和宽度都和目标的一致。
Matrix.ScaleToFit.START; //保持坐标变换前矩形的长宽比, 并最大限度的填充变换后的矩形。至少保持一边
Matrix.ScaleToFit.END; //保持坐标变换前矩形的长宽比, 并最大限度的填充变换后的矩形。至少保持一边

```



setPolyToPoly

77、setPolyToPoly的作用

通过指定 0~4个点、原始坐标、变换后坐标，来获得一个 变换矩阵(存入Matrix)

```
matrix.setPolyToPoly(
    float[] src, //The array of src [x,y] pairs (points)
    int srcIndex, //Index of the first pair of src values
    float[] dst, //The array of dst [x,y] pairs (points)
    int dstIndex, //Index of the first pair of dst values
    int pointCount); //The number of pairs/points to be used. Must be [0..4]
```

78、setPolyToPoly实例:1点-实现平移效果

```
// 1、源点
float[] src = {0, 0};
// 2、目标点
int DX = 300;
float[] dst = {0 + DX, 0 + DX};
// 3、matrix设置为该变换(平移)的矩阵
matrix.setPolyToPoly(src, 0, dst, 0, 1);

// 4、使用该Matrix进行平移操作
canvas.drawBitmap(bitmap, matrix, paint);
```

79、setPolyToPoly实例:2点-实现旋转或缩放移效果

```
//旋转
int bw = bitmap.getWidth();
int bh = bitmap.getHeight();
float[] src = {bw / 2, bh / 2, bw, 0};
float[] dst = {bw / 2, bh / 2, (bw / 2) + bh / 2, (bh / 2) + bw / 2};
matrix.setPolyToPoly(src, 0, dst, 0, 2);
canvas.drawBitmap(bitmap, matrix, paint);
```

第一个点是圆心，第二个点从右上角旋转到顺时针旋转90度后的位置

80、setPolyToPoly实例:3点-实现错切效果

```
Matrix matrix = new Matrix();
int bw = bitmap.getWidth();
int bh = bitmap.getHeight();
float[] src = {0, 0, 0, bh, bw, bh};
float[] dst = {0, 0, 200, bh, bw + 200, bh};
matrix.setPolyToPoly(src, 0, dst, 0, 3);
canvas.drawBitmap(bitmap, matrix, paint);
```

指定3个顶点，一个固定，另外两个移动。

81、setPolyToPoly实例:4点-实现透视效果(观察角度变换)

```
Matrix matrix = new Matrix();
int bw = bitmap.getWidth();
int bh = bitmap.getHeight();
float[] src = {0, 0, 0, bh, bw, bh, bw, 0};
int DX = 100;
float[] dst = {0 + DX, 0, 0, bh, bw, bh, bw - DX, 0};
matrix.setPolyToPoly(src, 0, dst, 0, 4);
canvas.drawBitmap(bitmap, matrix, paint);
```

收拢左上角和右上角两个顶点，达到3D倾斜效果。

mapXXX

82、mapPoints

```
// src中的点进行矩阵变换后(通过matrix)，会得到新的点，将这些新点存在dst数组中。
matrix.mapPoints(float[] dst, float[] src);
matrix.mapPoints(float[] dst, int dstIndex, float[] src, int srcIndex, int pointCount);
// pts数组中的点进行matrix变换后，将结果存回到pts数组中。
matrix.mapPoints(float[] pts);
```

83、mapVectors

1. 与 `mapPoints`同理，只是作用于 向量
2. 因为 向量 平移后依旧相等，因此 平移类`matrix` 没有实际影响。

```
public void mapVectors(float[] dst, int dstIndex, float[] src, int srcIndex,int vectorCount)
public void mapVectors(float[] dst, float[] src)
public void mapVectors(float[] vecs)
```

84、mapRect

1. 通过 `matrix` 对 矩阵`src` 进行矩阵变换，最终结果保存到 矩阵`dst` 中
2. 返回值 = 调用 `rectStaysRect()`

```
public boolean mapRect(RectF dst, RectF src)
public boolean mapRect(RectF rect)
```

95、mapRadius

```
public float mapRadius(float radius)
```

返回一个圆圈半径的平均值，将`matrix`作用于一个指定`radius`半径的圆，随后返回的平均半径。

贝塞尔曲线

基础与原理

86、贝塞尔曲线的由来

贝塞尔曲线于 1962年，由法国工程师皮埃尔·贝塞尔（Pierre Bézier）所发表，他运用贝塞尔曲线来为汽车的主体进行设计。

87、贝塞尔曲线的作用(The Bézier Curves)

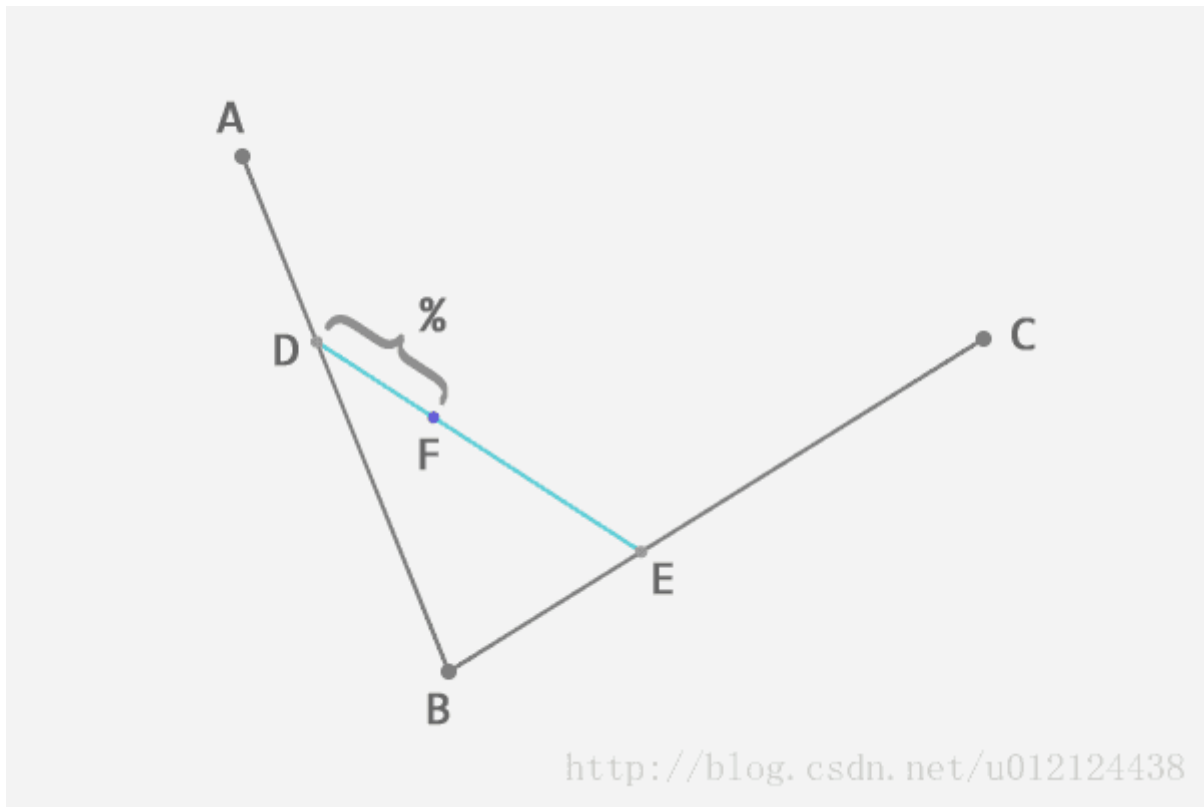
1. 是一种在计算机图形学中相当重要的参数曲线（2D，3D的称为曲面）。

88、贝塞尔曲线的绘制原理

A为起始点，C为结束点，B为控制点。

AB间有一点D，逐渐靠近B点。

任意时刻， $AD:DB=BE:EC$ ，此时连接DE,同理最终确定点 F 。无数个F相连接，最终绘制出从A点到C点的贝塞尔曲线。



$$AD:DB = BE:EC = DF:FE$$

89、贝塞尔曲线的公式推理

实际使用

90、Canvas中绘制贝塞尔曲线

91、SVG中绘制贝塞尔曲线

Region

构造方法

92、Region的构造方法

```
// 1. 无参数
Region region = new Region();
// 2. 由region对象构造
Region region = new Region(oldRegion);
// 3. 由Rect对象构造(矩形)
Rect rect = new Rect(left, top, right, bottom);
Region region = new Region(rect);
// 4. 用顶点坐标构造(矩形)
Region region = new Region(left, top, right, bottom);
```

替换方法

93、Region的替换方法

```
//1. 清空
region.setEmpty();
//2. 替换为oldRegion指定的区域
region.set(oldRegion);
//3. 替换为Rect指定的区域
region.set(oldRect);
//3. 替换为四顶点坐标指定的矩形区域
region.set(left, top, right, bottom);
//4. 替换为Path和Region两个区域的交集
region.setPath(path, region);
```

RegionIterator

94、RegionIterator的作用

1. 能将 Region区域 拆分为 一个个矩形(rectangles)
2. RegionIterator 会复制 Region , 因此后续对 Region 的操作不会影响到 iterator .

95、Region使用实例

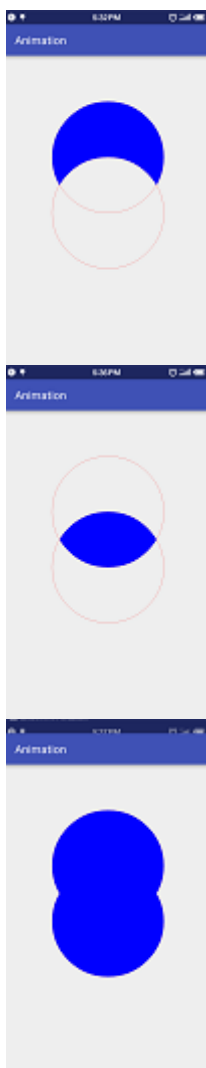
```
// 1.画笔设置
Paint mPaint = new Paint();
mPaint.setStyle(Paint.Style.STROKE);
mPaint.setStrokeWidth(3);
// 2.圆形Path
Path path = new Path();
path.addCircle(500, 500, 300, Path.Direction.CW);
// 3.Region与Path取交集
Region region = new Region();
region.setPath(path, new Region(0, 0, 1000, 1000));
// 4.迭代器将其拆分为矩形
RegionIterator iterator = new RegionIterator(region);
Rect rect = new Rect();
// 5.遍历绘制矩形
while (iterator.next(rect)) {
    canvas.drawRect(rect, mPaint);
}
```

区域操作

96、Region的区域操作

```
// 1. 取并集
region.union(rect);
// 2. 与目标区域进行操作(交、并、补、异并、反转补、后者区域替代前者)
region.op(otherRect, Region.Op.DIFFERENCE);
region.op(left, top, right, bottom, Region.Op.DIFFERENCE);
region.op(otherRegion, Region.Op.DIFFERENCE);
region.op(otherRect, otherRegion, Region.Op.DIFFERENCE);

// 3. Op的操作含义
DIFFERENCE(0), //取补集
INTERSECT(1), // 交集
UNION(2),      //并集
XOR(3),        //异并集
REVERSE_DIFFERENCE(4), //反转补集
REPLACE(5);    //后者区域替代前者
```





97、Region区域操作的实例

```

mPaint.setStyle(Paint.Style.STROKE);
mPaint.setStrokeWidth(1);

Path path1 = new Path();
Path path2 = new Path();
path1.addCircle(500, 500, 300, Path.Direction.CW);
path2.addCircle(500, 800, 300, Path.Direction.CW);

mPaint.setColor(Color.RED);
canvas.drawPath(path1,mPaint);
canvas.drawPath(path2,mPaint);

Region region = new Region();
Region region2 = new Region();
region.setPath(path1, new Region(0, 0, 1500, 1500));
region2.setPath(path2, new Region(0, 0, 1500, 1500));

region.op(region2, Region.Op.REPLACE);

//绘制
mPaint.setColor(Color.BLUE);
mPaint.setStyle(Paint.Style.FILL);
RegionIterator iterator = new RegionIterator(region);
Rect rect = new Rect();
while (iterator.next(rect)) {
    canvas.drawRect(rect, mPaint);
}

```

Picture

98、Picture的作用

1. A Picture records drawing calls (via the canvas returned by beginRecording)
2. 直接用Picture进行绘制的性能比 API调用 更高,

99、Picture相关API

```

picture.getHeight(); //高
picture.getWidth(); //宽
picture.beginRecording(width, height); //开始录制，返回canvas，后续的绘制在该Canvas上
picture.endRecording(); //结束录制
picture.draw(canvas); //将录制的内容绘制到canvas上

```

100、Picture的录制

```
Picture mPicture = new Picture();

private void recoding(){
    // 1、开始录制，获取到canvas
    Canvas canvas = mPicture.beginRecording(500, 500);
    // 2、用canvas进行绘制
    canvas.drawCircle(0, 0, 100, new Paint());
    // 3、结束录制
    mPicture.endRecording();
}
```

101、Picture的绘制方法(三种)

- 1-Picture的draw方法
- 2-Canvas的drawPicture方法
- 3-PictureDrawable的draw方法(进行包装)

```
// 1、Picture
mPicture.draw(canvas);

// 2、Canvas
canvas.drawPicture(mPicture);

// 3、PictureDrawable
PictureDrawable drawable = new PictureDrawable(mPicture);
// 设置绘制区域
drawable.setBounds(0, 0, 250, mPicture.getHeight());
drawable.draw(canvas);
```

参考资料

1. [Region区域](#)
2. [matrix 最全方法详解与进阶（完整篇）](#)
3. [layer详细讲解](#)
4. [贝塞尔曲线-公式推导](#)