Path特效之PathMeasure打造万能路径动效

标签： Path path特效 PathMeasure android动画 android动效 2015-07-26 16:10 3201人阅读 评论(5) 收藏 举报

分类： android动效篇（12）

我正在参加 CSDN 2015 博客之星评选 感恩分享活动，如果觉得文章还不错，请投个票鼓励下吧：
http://vote.blog.csdn.net/blogstar2015/candidate?username=tianjian4592

前面两篇文章主要讲解了 Path 的概念和基本使用，今天我们一起利用 Path 做个比较实用的小例子；

上一篇我们使用 Path 绘制了一个小桃心，我们这一篇继续围绕着这个小桃心进行展开：

如果对这个桃心绘制有问题或有兴趣的同学，可以链接到 Path相关方法讲解(二)，此时我们的需求是这样的：

假定我们现在是一个婚恋产品，有一个“心动”的功能，用户点击“心动”按钮的时候，有一个光点快速的沿着桃心转一圈，然后整个桃心泛起光晕！

针对这个需求，很多人可能会想到以下方案：

不就一个光点沿着桃心跑一圈么，既然桃心是使用贝塞尔曲线画出来的，那么我们就可以用对应的函数模拟出这条曲线，然后算出对应位置上的点，不断将光点绘制到对应的位置上！

这个思路当然没有问题，但我们还有相对简单的方式，那就是使用 PathMeasure:

我们主要使用它两个方法：

1.getLength() - 获取路径的长度

2.getPosTan(float distance, float pos[],float tan[]) — path 为 null ，返回 false

distance 为一个 0 — getLength() 之间的值，根据这个值 PathMeasure 会计算出当前点的坐标封装到 pos 中；

上面这句话我们可以这么来理解，不管实际 Path 多么的复杂，PathMeasure 都相当于做了一个事情，就是把 Path “拉直”，然后给了我们一个接口(getLength)告诉我们path的总长度，然后我们想要知道具体某一点的坐标，只需要用相对的distance去取即可，这样就省去了自己用函数模拟path，然后计算获取点坐标的过程；

接下来，我们用代码实现这一效果：

我们先创建一个 PathMeasure ，并将创建好的 path 作为参数传入

```
[html]
mPath = new Path();
mPath.moveTo(START_POINT[0], START_POINT[1]);
mPath.quadTo(RIGHT_CONTROL_POINT[0], RIGHT_CONTROL_POINT[1], BOTTOM_POINT[0],
        BOTTOM_POINT[1]);
mPath.quadTo(LEFT_CONTROL_POINT[0], LEFT_CONTROL_POINT[1], START_POINT[0], START_POINT[1]);
mPathMeasure = new PathMeasure(mPath, true);
```

然后用一个数组纪录点的坐标：

```
[html]
private float[] mCurrentPosition = new float[2];
```

向外暴露一个开启动效的接口：

```
[html]
// 开启路径动画
public void startPathAnim(long duration) {
    // 0 — getLength()
    ValueAnimator valueAnimator = ValueAnimator.ofFloat(0, mPathMeasure.getLength());
    Log.i(TAG, "measure length = " + mPathMeasure.getLength());
    valueAnimator.setDuration(duration);
    // 减速插值器
    valueAnimator.setInterpolator(new DecelerateInterpolator());
    valueAnimator.addUpdateListener(new AnimatorUpdateListener() {

        @Override
        public void onAnimationUpdate(ValueAnimator animation) {
            float value = (Float) animation.getAnimatedValue();
            // 获取当前点坐标封装到mCurrentPosition
            mPathMeasure.getPosTan(value, mCurrentPosition, null);
            postInvalidate();
        }
    });
    valueAnimator.start();
}
```

实时获取到当前点之后，将目标绘制到对应位置：

```
[html]
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    canvas.drawColor(Color.WHITE);
    canvas.drawPath(mPath, mPaint);

    canvas.drawCircle(RIGHT_CONTROL_POINT[0], RIGHT_CONTROL_POINT[1], 5, mPaint);
    canvas.drawCircle(LEFT_CONTROL_POINT[0], LEFT_CONTROL_POINT[1], 5, mPaint);

    // 绘制对应目标
    canvas.drawCircle(mCurrentPosition[0], mCurrentPosition[1], 10, mPaint);
}
```

到这里目标环绕 path 的效果就ok了，不管这条路径简单也好，复杂也罢，我们都可以如此简单的完成对应的效果，而不需要自己用简单或复杂函数模拟求解了；

完成了一步，自己提的需求还有一点就是光晕的问题，这个东西如何是好呢？切图？！ 不需要，android 已经给我们提供了一个好用的东西 MaskFilter ，后面我就不做了，大家有兴趣自己做的玩玩，只需要注意一点，MaskFilter 不支持硬件加速，记得关掉！

好了，PathMeasure 看似很简单，但着实很有用，有了它，再结合上 Path 、Shader、ColorMatrix 等利器，我们已经可以做出很多酷炫的效果了！

最后，完整的代码献上，请笑纳：

```
[html]
public class DynamicHeartView extends View {

    private static final String TAG = "DynamicHeartView";
    private static final int PATH_WIDTH = 2;
    // 起始点
    private static final int[] START_POINT = new int[] {
            300, 270
    };
    // 爱心下端点
    private static final int[] BOTTOM_POINT = new int[] {
            300, 400
    };
    // 左侧控制点
    private static final int[] LEFT_CONTROL_POINT = new int[] {
            450, 200
    };
    // 右侧控制点
    private static final int[] RIGHT_CONTROL_POINT = new int[] {
            150, 200
    };

    private PathMeasure mPathMeasure;
    private Paint mPaint;
    private Path mPath;
    private float[] mCurrentPosition = new float[2];

    public DynamicHeartView(Context context) {
        super(context);
        init();
    }
```

```
31.
32.     private void init() {
33.         mPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
34.         mPaint.setStyle(Style.STROKE);
35.         mPaint.setStrokeWidth(PATH_WIDTH);
36.         mPaint.setColor(Color.RED);
37.
38.         mPath = new Path();
39.         mPath.moveTo(START_POINT[0], START_POINT[1]);
40.         mPath.quadTo(RIGHT_CONTROL_POINT[0], RIGHT_CONTROL_POINT[1], BOTTOM_POINT[0],
41.                 BOTTOM_POINT[1]);
42.         mPath.quadTo(LEFT_CONTROL_POINT[0], LEFT_CONTROL_POINT[1], START_POINT[0], START_P(
43.
44.         mPathMeasure = new PathMeasure(mPath, true);
45.         mCurrentPosition = new float[2];
46.     }
47.
48.     @Override
49.     protected void onDraw(Canvas canvas) {
50.         super.onDraw(canvas);
51.         canvas.drawColor(Color.WHITE);
52.         canvas.drawPath(mPath, mPaint);
53.
54.         canvas.drawCircle(RIGHT_CONTROL_POINT[0], RIGHT_CONTROL_POINT[1], 5, mPaint);
55.         canvas.drawCircle(LEFT_CONTROL_POINT[0], LEFT_CONTROL_POINT[1], 5, mPaint);
56.
57.         // 绘制对应目标
58.         canvas.drawCircle(mCurrentPosition[0], mCurrentPosition[1], 10, mPaint);
59.     }
60.
61.     // 开启路径动画
62.     public void startPathAnim(long duration) {
63.         // 0 — getLength()
64.         ValueAnimator valueAnimator = ValueAnimator.ofFloat(0, mPathMeasure.getLength());
65.         Log.i(TAG, "measure length = " + mPathMeasure.getLength());
66.         valueAnimator.setDuration(duration);
67.         // 减速插值器
68.         valueAnimator.setInterpolator(new DecelerateInterpolator());
69.         valueAnimator.addUpdateListener(new AnimatorUpdateListener() {
70.
71.             @Override
72.             public void onAnimationUpdate(ValueAnimator animation) {
73.                 float value = (Float) animation.getAnimatedValue();
74.                 // 获取当前点坐标封装到mCurrentPosition
75.                 mPathMeasure.getPosTan(value, mCurrentPosition, null);
76.                 postInvalidate();
77.             }
78.         });
79.         valueAnimator.start();
80.
81.     }
82. }
```

以上代码的效果如下，其余效果大家自行补充：

PathMeasure打造万能路径特效 — 源码下载