【Android开源项目解析】仿支付宝付款成功及"天女散花"效果实现——看PathMeasure大展身手

标签： path pathMeatur android

2015-09-15 21:35 3353人阅读 评论(5) 收藏 举报

分类： Android开源项目解析（3） Android自定义控件使用（16）

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录(?) [+]

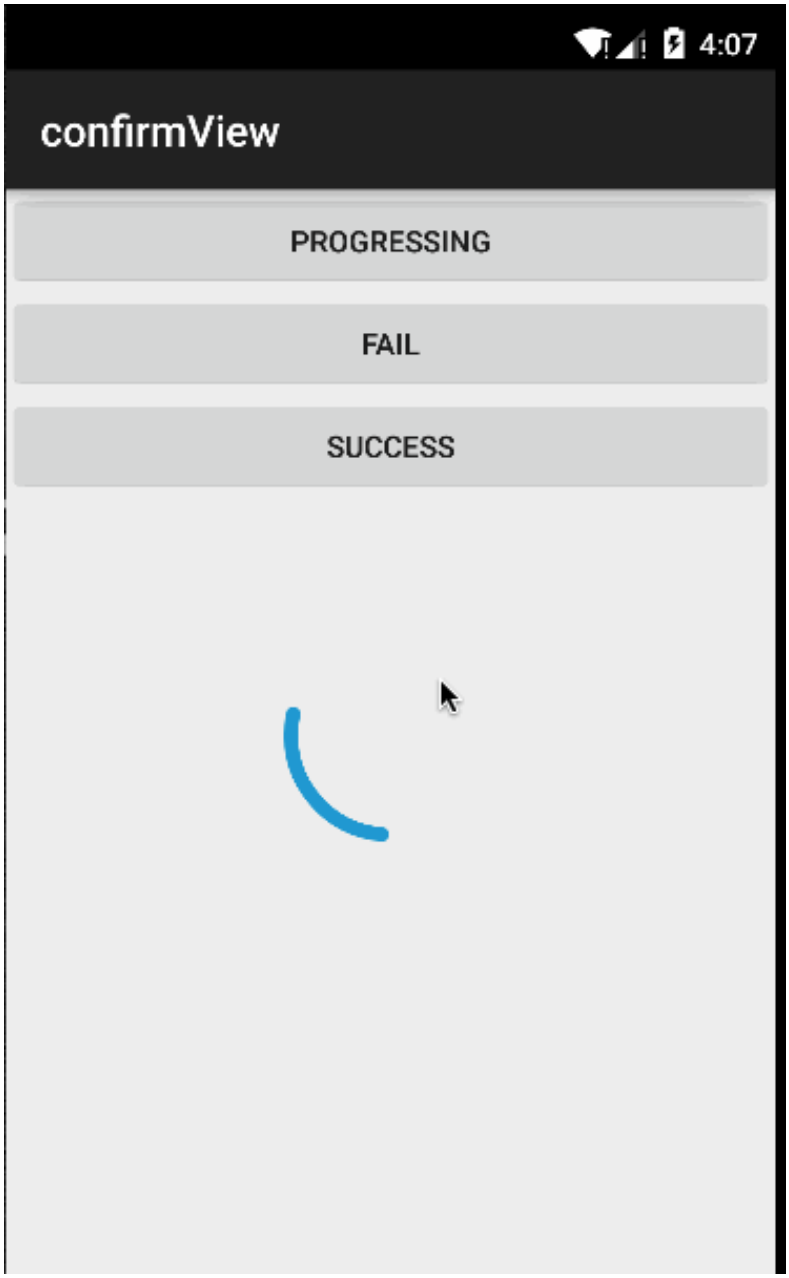
> 话说，在前面两篇文章中，我们学习了BitmapShader、Path的基本使用，那么这一篇文章，咱们接着来学习一下PathMeasure的用法。什么，你没听说过PathMeasure？那你就要OUT咯~

项目效果图

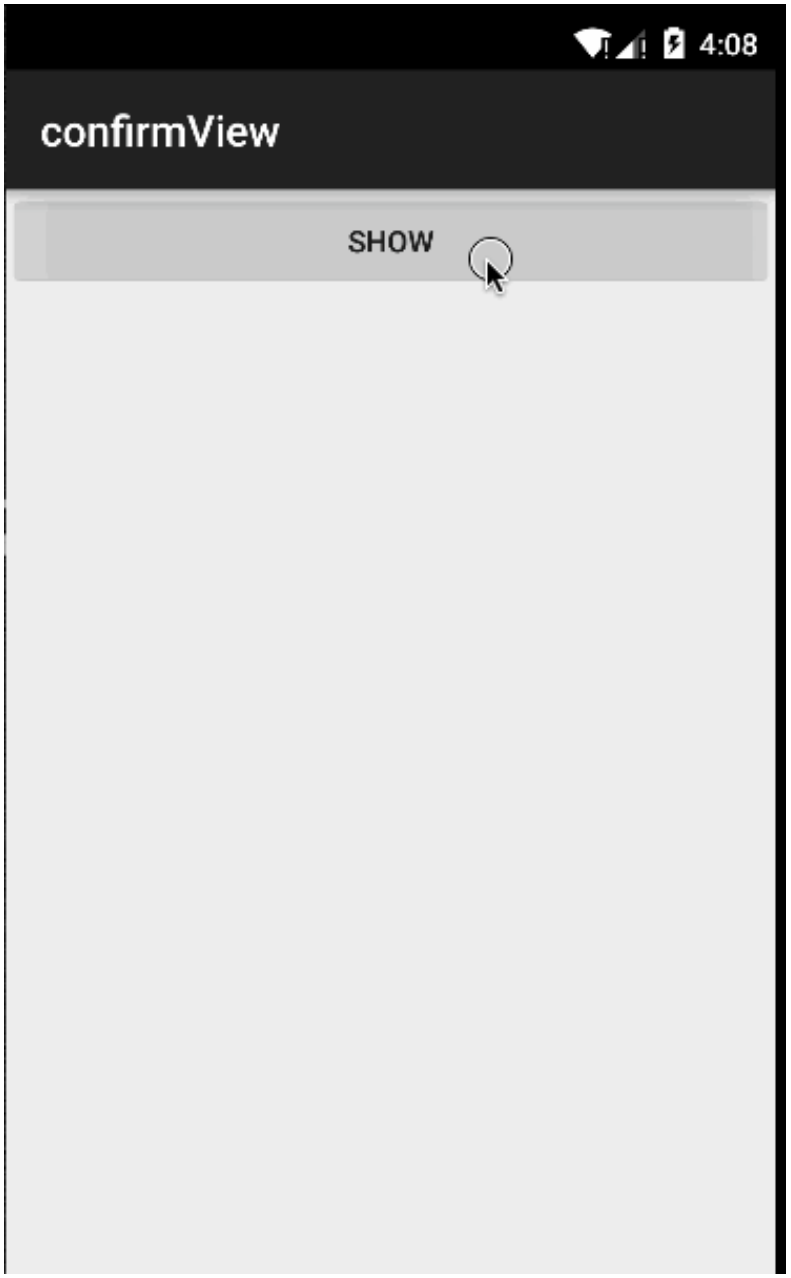
废话不多说，在开始讲解之前，先看下最终实现的效果。

效果一：

仿支付宝支付成功效果



效果二：



这两个项目都是使用Path和PathMeature配合完成的，由其他项目改造而来

项目一是七叔写的，我对代码进行了大量改造。

项目二是不小心搜到的，然后进行了改造，原文请戳这里

本文代码请到这里下载

PathMeasure介绍

PathMeasure这个类确实是不太常见的，关于这个类的介绍也是甚少，那么这个类是用来干嘛的呢？主要其实是配合Path，来计算Path里面点的坐标的，或者是给一个范围，来截取Path其中的一部分的。

这么说，你肯定也迷糊，咱们先简单看一下有哪些方法，然后根据案例来进行讲解更好一些。

构造方法有两个，很好理解，不多解释。

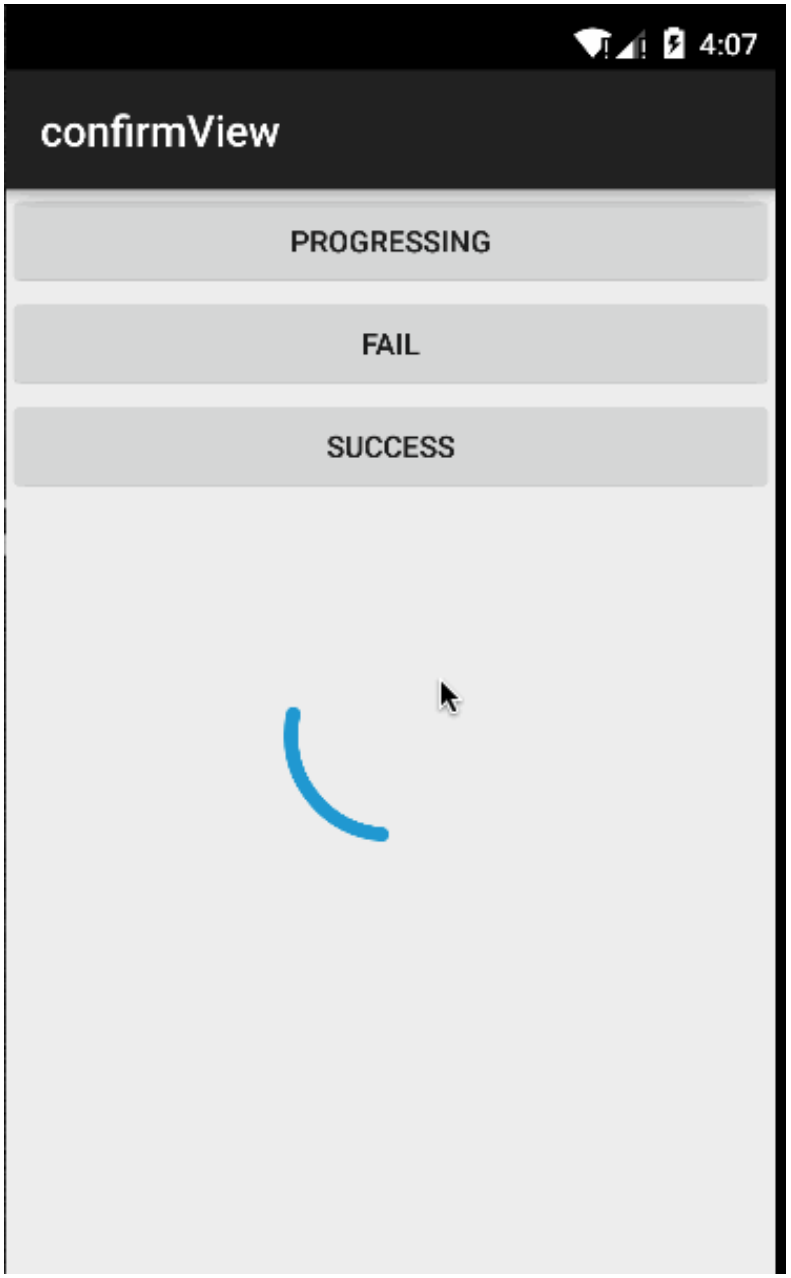
```
PathMeasure()
PathMeasure(Path path, boolean forceClosed)
```

重点看下常用方法：

- float getLength() 返回当前contour(解释为轮廓不太恰当，我觉得更像是笔画)的长度，也就是这一个Path有多长
- boolean getPosTan(float distance, float[] pos, float[] tan) 传入一个距离distance(0<=distance<=getLength())，然后会计算当前距离的坐标点和切线，注意，pos会自动填充上坐标，这个方法很重要
- boolean getSegment(float startD, float stopD, Path dst, boolean startWithMoveTo) 传入一个开始和结束距离，然后会返回介于这之间的Path，在这里就是dst，他会被填充上内容，这个方法很重要
- boolean nextContour() 移动到下一个笔画，如果你的Path是由多个笔画组成的话，那么就可以使用这个方法
- void setPath(Path path, boolean forceClosed)这个方法也比较重要，用来设置新的Path对象的，算是对第一个构造函数的一个补充

仿支付宝实现原理解析

下面，我将介绍一下如何实现下面的这个效果



首先分析需求：

- 需要有三种状态：加载中，成功，失败
- 加载中时，需要不断更换颜色
- 加载中状态时，圆弧要不断的变换长度和位置
- 成功状态和失败状态，需要把√和×一笔一划的画出来

OK，基本就是这些需求，那么对应着需求，咱们看一下解决方案

- 有三种状态好说，用静态常量或者是枚举类型进行区分
- 不断变换颜色也好说，只要改变Paint的颜色就可以啦
- 不断的变化长度和位置，从效果图上可以看出来，我们需要画一段圆弧，那就要用下面的drawArc()，需要知道范围，起始角度和绘制角度，由于需要不断的变化长度，因此就需要用Animator，具体实现一会详谈

```
1 Canvas.drawArc(RectF oval, float startAngle, float sweepAngle, boolean useCenter,Paint
```

- 需要画出来形状，其实就是一些线段，那么就需要用Path了，但是如何能一笔一划的效果呢？那就要靠PathMeasure啦

下面开始讲解代码实现，最好参照着源代码看下面的文章。

首先看怎么用ConfirmView呢？很简单，只需要调用animatedWithState()然后传入一个枚举类型即可

```
confirmView.animatedWithState(ConfirmView.State.Progressing);
```

这个枚举类型在类的内部，代表三种状态

```
public enum State {
        Success, Fail, Progressing
    }
```

再看构造函数，很简单，只是进行了变量的初始化，这些变量的具体作用，我将在下面用到的时候重点介绍

```
public ConfirmView(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);

        mSuccessPath = new Path();
        mPathMeasure = new PathMeasure(mSuccessPath, false);
        mRenderPaths = new ArrayList<>();

        mPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
        mPaint.setStyle(Paint.Style.STROKE);
        mPaint.setColor(0xFF0099CC);
        mPaint.setStrokeWidth(STROKEN_WIDTH);
        mPaint.setStrokeCap(Paint.Cap.ROUND);

        oval = new RectF();
    }
```

那么调用了animatedWithState()之后，进行了什么操作呢？

```
public void animatedWithState(State state) {

        if (mCurrentState != state) {
            mCurrentState = state;
            if (mPhareAnimator != null && mPhareAnimator.isRunning()) {
                stopPhareAnimation();
            }
            switch (state) {
                case Fail:
                case Success:
                    updatePath();
```

```
                    if (mCircleAnimator != null && mCircleAnimator.isRunning()) {
                        mCircleAngle = (Float) mCircleAnimator.getAnimatedValue();
                        mCircleAnimator.end();
                    }

                    if ((mStartAngleAnimator == null || !mStartAngleAnimator.isRunning(
                            (mEndAngleAnimator == null || !mEndAngleAnimator.isRunning(
                        mStartAngle = 360;
                        mEndAngle = 0;
                        startPhareAnimation();
                    }

                    break;
                case Progressing:
                    mCircleAngle = 0;
                    startCircleAnimation();
                    break;
            }
        }

    }
```

结合着上面的代码，我简单解释一下。

首先进行重复性的判断，如果当前所处的状态与要改变的状态相同则不进行操作。

接下来，对动画状态进行了判断，mPhareAnimator是用来实现√和×的动画绘制效果的，如果正在运行，则停掉。

再往下的一个switch则是开始真正的操作了，updatePath()是更新Path，一会重点看下，mCircleAnimator这个则是实现外部弧形的偏移量的控制的，现在看不明白也没事，重点看下下面的代码，当mStartAngleAnimator和mEndAngleAnimator都不在运行状态的时候(这两个Animator是为了控制外部弧形的起点和终点的)，会进入下面的代码，

```
mStartAngle = 360;
mEndAngle = 0;
startPhareAnimation();
```

mStartAngle和mEndAngle分别代表起点转过的角度和终点转过的角度，然后就startPhareAnimation()，这个时候，真正的绘制√和×的动画才开始执行。

如果是Progressing呢，则执行下面的代码，重置mCircleAngle，startCircleAnimation()这个方法是绘制外部的弧形的动画

```
mCircleAngle = 0;
startCircleAnimation();
```

至此，咱们知道了传入不同状态的枚举类型会进行什么操作，下面，开始看真正的操作。

咱先看一个简单的，就是startCircleAnimation()到底做了什么。

前面说过，这个方法是为了绘制加载中状态时，外部不断变化的彩色弧形的，下面是代码实现

```
public void startCircleAnimation() {
        if (mCircleAnimator == null || mStartAngleAnimator == null || mEndAngleAnimator
            initAngleAnimation();
        }
        mStartAngleAnimator.setDuration(NORMAL_ANGLE_ANIMATION_DURATION);
    mEndAngleAnimator.setDuration(NORMAL_ANGLE_ANIMATION_DURATION);
        mCircleAnimator.setDuration(NORMAL_CIRCLE_ANIMATION_DURATION);
        mStartAngleAnimator.start();
        mEndAngleAnimator.start();
        mCircleAnimator.start();
    }
```

首先前面的if语句是为空判断，从而进行初始化的操作，后面则是简单的设置动画的持续时间和开启动画。这里一共出现了三个动画，完成外部弧形的效果控制

- mStartAngleAnimator 控制圆弧起点
- mEndAngleAnimator 控制圆弧终点
- mCircleAnimator 控制圆弧的整体偏移量

这么说，你可能还是不很明白，没关系，咱们一点点的看代码，首先，咱们看在初始化的时候，到底做了什么操作，也就是initAngleAnimation()。

```
 private void initAngleAnimation() {

        mStartAngleAnimator = ValueAnimator.ofFloat(0.0F, 1.0F);
        mEndAngleAnimator = ValueAnimator.ofFloat(0.0F, 1.0F);
        mCircleAnimator = ValueAnimator.ofFloat(0.0F, 1.0F);

        mStartAngleAnimator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener(
            @Override
            public void onAnimationUpdate(ValueAnimator animation) {
                float value = (Float) animation.getAnimatedValue();
                setStartAngle(value);
            }
        });
        mEndAngleAnimator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener()
            @Override
            public void onAnimationUpdate(ValueAnimator animation) {
                float value = (Float) animation.getAnimatedValue();
```

```
18                setEndAngle(value);
19            }
20        });
21
22        mStartAngleAnimator.addListener(new Animator.AnimatorListener() {
23            @Override
24            public void onAnimationStart(Animator animation) {
25
26                if (mCurrentState == State.Progressing) {
27                    if (mEndAngleAnimator != null) {
28                        new Handler().postDelayed(new Runnable() {
29                            @Override
30                            public void run() {
31                                mEndAngleAnimator.start();
32                            }
33                        }, 400L);
34                    }
35                }
36            }
37
38            @Override
39            public void onAnimationEnd(Animator animation) {
40                if (mCurrentState != State.Progressing && mEndAngleAnimator != null &&
41                    startPhareAnimation();
42                }
43            }
44
45            @Override
46            public void onAnimationCancel(Animator animation) {
47            }
48
49            @Override
50            public void onAnimationRepeat(Animator animation) {
51            }
52        });
53
54        mEndAngleAnimator.addListener(new Animator.AnimatorListener() {
55            @Override
56            public void onAnimationStart(Animator animation) {
57
58            }
59
60            @Override
61            public void onAnimationEnd(Animator animation) {
62                if (mStartAngleAnimator != null) {
63                    if (mCurrentState != State.Progressing) {
64                        mStartAngleAnimator.setDuration(NORMAL_ANIMATION_DURATION);
65                    }
66                    colorCursor++;
```

```
                    if (colorCursor >= colors.length) colorCursor = 0;
                    mPaint.setColor(colors[colorCursor]);
                    mStartAngleAnimator.start();
                }
            }

            @Override
            public void onAnimationCancel(Animator animation) {
            }

            @Override
            public void onAnimationRepeat(Animator animation) {
            }
        });

        mCircleAnimator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
            @Override
            public void onAnimationUpdate(ValueAnimator animation) {
                float value = (float) animation.getAnimatedValue();
                setCircleAngle(value);
            }
        });


        mStartAngleAnimator.setInterpolator(new AccelerateDecelerateInterpolator());
        mEndAngleAnimator.setInterpolator(new AccelerateDecelerateInterpolator());


        mCircleAnimator.setInterpolator(new LinearInterpolator());
        mCircleAnimator.setRepeatCount(-1);
    }
```

这段代码虽然长，但是也没有太大的难度，无非就是进行了初始化操作，ValueAnimator的范围是0-1，这个在后面将用于计算角度。在值不断的更新的过程中，分别调用了下面这三个方法，更新一些值

setStartAngle(value);
setEndAngle(value);
setCircleAngle(value);

在这三个方法里面，都对成员变量进行了更新，并且！调用了invalidate()！看到这里是不是激动了，改变一次就重绘一次，这三个值肯定和弧形的动画效果有关啊！

```
    private void setStartAngle(float startAngle) {
        this.mStartAngle = startAngle;
        invalidate();
    }

```

```
    private void setEndAngle(float endAngle) {
        this.mEndAngle = endAngle;
        invalidate();
    }

    private void setCircleAngle(float circleAngle) {
        this.mCircleAngle = circleAngle;
        invalidate();
    }
```

咱知道了这个，先不着急去看onDraw()，仔细看下动画的执行顺序。

在mStartAngleAnimator执行之后，调用了下面的方法，这当然很简单，就是说，mStartAngleAnimator执行了400毫秒之后，mEndAngleAnimator才会执行，而且插值器设置的是AccelerateDecelerateInterpolator，为啥呢？很简单，因为只有这样，才能做出弧形长度先长后短的效果呀~

```
new Handler().postDelayed(new Runnable() {
       @Override
        public void run() {
            mEndAngleAnimator.start();
        }
       }, 400L);
```

而在mEndAngleAnimator执行结束之后，会调用下面的代码

```
if (mStartAngleAnimator != null) {
    if (mCurrentState != State.Progressing) {
        mStartAngleAnimator.setDuration(NORMAL_ANIMATION_DURATION);
    }
    colorCursor++;
    if (colorCursor >= colors.length) colorCursor = 0;
    mPaint.setColor(colors[colorCursor]);
    mStartAngleAnimator.start();
}
```

在这个设置mStartAngleAnimator的动画时间，是为了画√或者是×的时候快一些效果更流畅。下面的代码很简单了吧，改变画笔颜色，然后mStartAngleAnimator又开启啦！这就是为啥一直转啊转的原因。

但是说到这里，咱们还没看onDraw（ ）做了什么呢！

```
 @Override
    public void onDraw(Canvas canvas) {
        super.onDraw(canvas);

        switch (mCurrentState) {
            case Fail:
```

```
                for (int i = 0; i < PATH_SIZE_TWO; i++) {
                    Path p = mRenderPaths.get(i);
                    if (p != null) {
                        canvas.drawPath(p, mPaint);
                    }
                }
                drawCircle(canvas);
                break;
            case Success:
                Path p = mRenderPaths.get(0);
                if (p != null) {
                    canvas.drawPath(p, mPaint);
                }
                drawCircle(canvas);
                break;
            case Progressing:
                drawCircle(canvas);
                break;
        }

    }
```

咱先看Progressing分支里面的drawCircle(canvas)，其他的先不要管

```
private void drawCircle(Canvas canvas) {
        float offsetAngle = mCircleAngle * 360;
        float startAngle = mEndAngle * 360;
        float sweepAngle = mStartAngle * 360;

        if (startAngle == 360)
            startAngle = 0;
        sweepAngle = sweepAngle - startAngle;
        startAngle += offsetAngle;

        if (sweepAngle < 0)
            sweepAngle = 1;

        canvas.drawArc(oval, startAngle, sweepAngle, false, mPaint);
    }
```

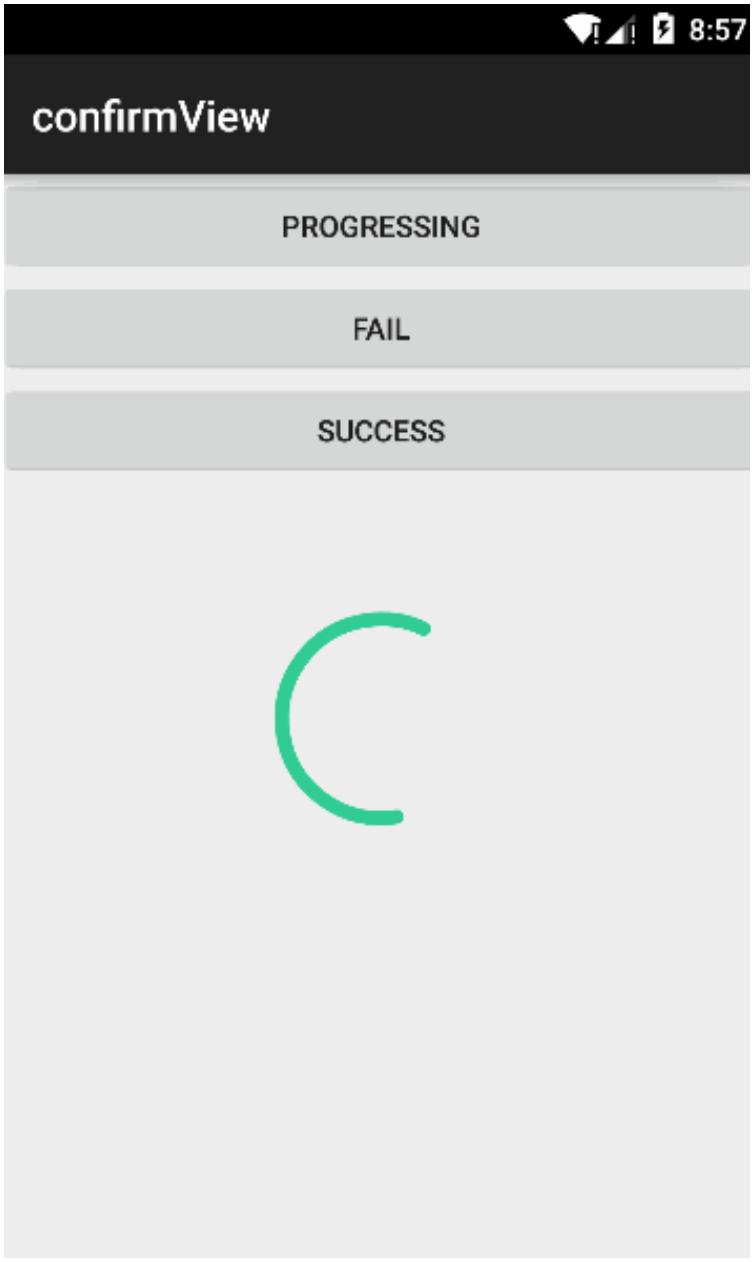
是的，上面这段代码就是绘制不断变幻的环的代码咯

float startAngle = mEndAngle * 360;是计算终点的位置，有人会感到奇怪，为啥终点的位置叫startAngle啊！因为终点的位置就是开始绘制的位置，所以不要奇怪了。

sweepAngle = sweepAngle - startAngle;则是计算要画多少角度的弧线，因为起点先跑到前面的，所以减去终点的位置，就是旋转角度。

startAngle += offsetAngle;那么这句是干嘛的？这个就是所谓的偏移量，为了要实现更随性的从非固定点开始结束的效果。没听懂？我给你去掉你看下效果！

```
1
2     private void drawCircle(Canvas canvas) {
3         float offsetAngle = mCircleAngle * 360;
4         float startAngle = mEndAngle * 360;
5         float sweepAngle = mStartAngle * 360;
6 
7         if (startAngle == 360)
8             startAngle = 0;
9         sweepAngle = sweepAngle - startAngle;
10 //         startAngle += offsetAngle;
11 
12        if (sweepAngle < 0)
13            sweepAngle = 1;
14 
15        canvas.drawArc(oval, startAngle, sweepAngle, false, mPaint);
16    }
```



这下子明白了吧，去掉漂移量效果就没有之前那么随性了~

---

ok，关于弧线的问题就说这么多，下面就要说咱们今天的主角PathMeasure了。

在前面的代码中，我们提到，成功和失败状态会执行updatePath()和startPhareAnimation()，那么到底做了些什么呢？

```
1  private void updatePath() {
2
3          int offset = (int) (mSignRadius * 0.15F);
4          mRenderPaths.clear();
5
6          switch (mCurrentState) {
7              case Success:
8                  mSuccessPath.reset();
9                  mSuccessPath.moveTo(mCenterX - mSignRadius, mCenterY + offset);
```

```
                mSuccessPath.lineTo(mCenterX - offset, mCenterY + mSignRadius - offset)
                mSuccessPath.lineTo(mCenterX + mSignRadius, mCenterY - mSignRadius + of
                mRenderPaths.add(new Path());
                break;
            case Fail:
                mSuccessPath.reset();
                float failRadius = mSignRadius * 0.8F;
                mSuccessPath.moveTo(mCenterX - failRadius, mCenterY - failRadius);
                mSuccessPath.lineTo(mCenterX + failRadius, mCenterY + failRadius);
                mSuccessPath.moveTo(mCenterX + failRadius, mCenterY - failRadius);
                mSuccessPath.lineTo(mCenterX - failRadius, mCenterY + failRadius);
                for (int i = 0; i < PATH_SIZE_TWO; i++) {
                    mRenderPaths.add(new Path());
                }
                break;
            default:
                mSuccessPath.reset();
        }

        mPathMeasure.setPath(mSuccessPath, false);

    }
```

在updatePath()我们可以很清楚的看到，在这里初始化了mSuccessPath，通过moveTo()和lineTo()首先勾勒除了√和×的形状，至于这个坐标是怎么确定的，这个可以自己想法来，我就不介绍了。还要需要注意的是，Success中最后在mRenderPaths中添加了一个Path对象，而在Fail则添加了两个对象，这个其实是和要绘制的图形的笔画数有关的，×是两笔，所以是两个，这里添加的Path议会将用来纪录每一笔画的形状。

最后，咱们的主角终于现身了

```
 mPathMeasure.setPath(mSuccessPath, false);
```

调用完这个方法，会马上调用下面的方法

```
public void startPhareAnimation() {
        if (mPhareAnimator == null) {
            mPhareAnimator = ValueAnimator.ofFloat(0.0F, 1.0F);
            mPhareAnimator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener()
                @Override
                public void onAnimationUpdate(ValueAnimator animation) {
                    float value = (Float) animation.getAnimatedValue();
                    setPhare(value);
                }
            });

            mPhareAnimator.setDuration(NORMAL_ANIMATION_DURATION);
            mPhareAnimator.setInterpolator(new LinearInterpolator());
```

```
        }
        mPhare = 0;
        mPhareAnimator.start();
    }
```

其实也很简单，初始化了mPhareAnimator，然后开启动画，不断调用setPhare(value)，

```
private void setPhare(float phare) {
        mPhare = phare;
        updatePhare();
        invalidate();
    }
```

在这里updatePhare()，然后重绘界面，那么玄机应该都在updatePhare()了吧！

```
private void updatePhare() {

        if (mSuccessPath != null) {
            switch (mCurrentState) {
                case Success: {
                    if (mPathMeasure.getSegment(0, mPhare * mPathMeasure.getLength(), m
                        mRenderPaths.get(0).rLineTo(0, 0);
                    }
                }
                break;
                case Fail: {
                    //i = 0,画一半， i=1,画另一半
                    float seg = 1.0F / PATH_SIZE_TWO;

                    for (int i = 0; i < PATH_SIZE_TWO; i++) {
                        float offset = mPhare - seg * i;
                        offset = offset < 0 ? 0 : offset;
                        offset *= PATH_SIZE_TWO;
                        Log.d("i:" + i + ",seg:" + seg, "offset:" + offset + ", mPhare:
                        boolean success = mPathMeasure.getSegment(0, offset * mPathMeas

                        if (success) {
                            mRenderPaths.get(i).rLineTo(0, 0);
                        }
                        mPathMeasure.nextContour();
                    }
                    mPathMeasure.setPath(mSuccessPath, false);
                }
                break;
            }
        }
    }
```

在这里，一个很重要的方法调用了，那就是mPathMeasure.getSegment()

当Success的时候，会执行下面的代码。mPhare就是动画的百分比，从0到1，那么，下面的这段代码就很好理解了，这是为了根据动画的百分比，获取画出√的整个Path的一部分，然后把这部分，填充到了mRenderPaths.get(0)里面，这里面存放的就是在上面方法中添加进去的一个Path对象。mPhare不断的变化，我们就能获取到画整个√形状所需的所有Path对象，还记得这个方法之后是什么吗？invalidate()！所以，现在在onDraw()里面肯定用这Path对象，画出√的一部分，不断的更新从mPhare，不断绘制，从无到有，而出现了动画效果。

mRenderPaths.get(0).rLineTo(0, 0);这个代码则是为了在4.4以下不能绘制出图形BUG的解决方法，不要在意。

```
if (mPathMeasure.getSegment(0, mPhare * mPathMeasure.getLength(), mRenderPaths.get(0),
     mRenderPaths.get(0).rLineTo(0, 0);
}
```

不信咱们看下onDraw()，是不是！那么现在你应该知道×是怎么画出来的吧？

```
case Success:
Path p = mRenderPaths.get(0);
if (p != null) {
    canvas.drawPath(p, mPaint);
}
drawCircle(canvas);
break;
```

来来来，咱们看下代码！

```
 case Fail: {
   //i = 0,画一半，i=1,画另一半
     float seg = 1.0F / PATH_SIZE_TWO;
     for (int i = 0; i < PATH_SIZE_TWO; i++) {
         float offset = mPhare - seg * i;
         offset = offset < 0 ? 0 : offset;
         offset *= PATH_SIZE_TWO;
         Log.d("i:" + i + ",seg:" + seg, "offset:" + offset + ", mPhare:" + mPhare + ",
         boolean success = mPathMeasure.getSegment(0, offset * mPathMeasure.getLength()

         if (success) {
             mRenderPaths.get(i).rLineTo(0, 0);
         }

         mPathMeasure.nextContour();
     }
     mPathMeasure.setPath(mSuccessPath, false);
```

```
17  }
18  break;
```

与绘制√相比，因为×是两笔，所以有些小复杂，但是也不难，offset *= PATH_SIZE_TWO;是为了保证在mPhare从0-0.5过程中控制第一笔画，0.5-1则控制第二条笔画，你仔细看下代码，这样可以实现offset从0-1两次。由于×是两笔画，所以在i=0取到第一笔画的Path部分，存储在mRenderPaths的第一个Path之后，调用了mPathMeasure.nextContour();切换到下一笔画，再次完成相同的操作。

而由于PathMeasure只能往下找Contour，所以最后 mPathMeasure.setPath(mSuccessPath, false);回复到最初状态，然后我们看下onDraw()

```
1  for (int i = 0; i < PATH_SIZE_TWO; i++) {
2                  Path p = mRenderPaths.get(i);
3                  if (p != null) {
4                      canvas.drawPath(p, mPaint);
5                  }
6              }
7              drawCircle(canvas);
```

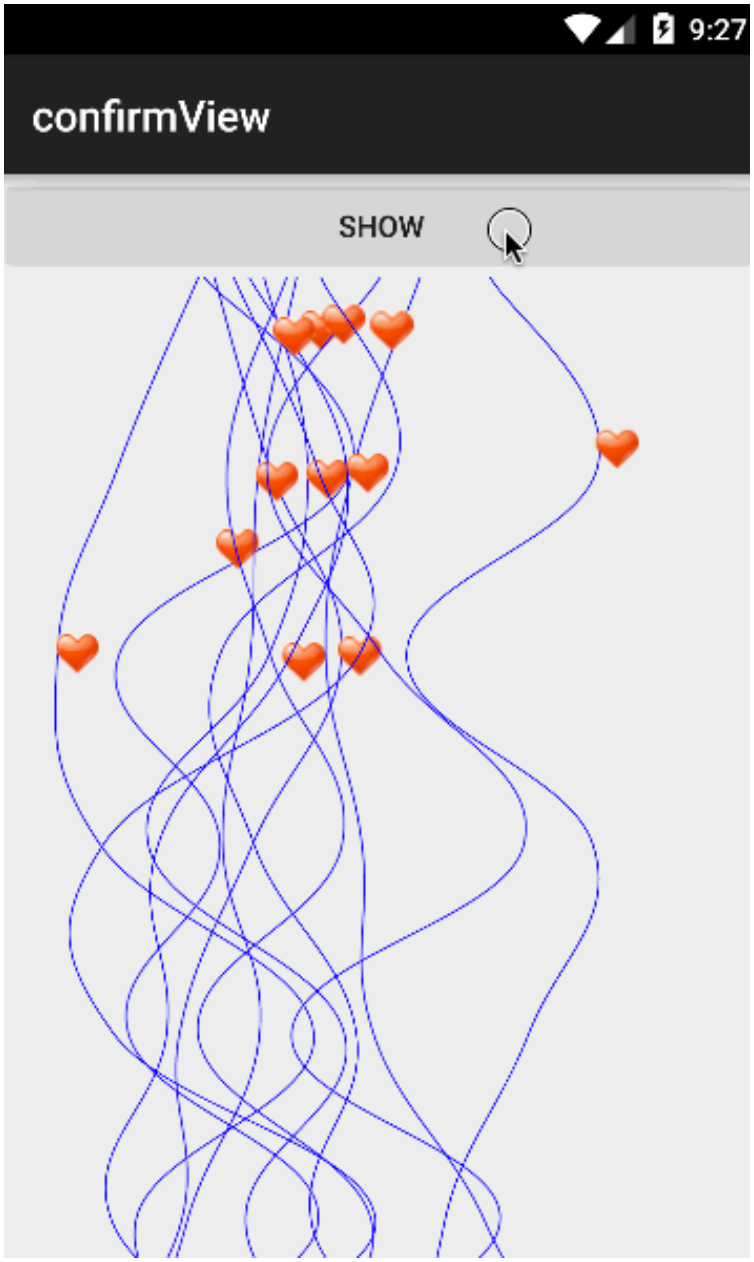
其实和Success差不多的，只不过是两个Path，画出两笔。

OK，到这里，这个效果就算是全部实现了，累死我了

"天女散花"实现效果解析

其实这个我并不打算详细讲，因为一通百通，多说无益，更多的东西需要你自己研究代码吸收，咱们就重点看下PathMeasure的用法。

其实这种效果实现的真相是这样滴



YES!就是一些Bitmap对象沿着Path路径移动！

那么和PathMeasure有啥关系呢？

看下onDraw()！

```
@Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        drawFllower(canvas, fllowers1);
        drawFllower(canvas, fllowers2);
        drawFllower(canvas, fllowers3);
    }
```

OK，再看下drawFllower()

```
private void drawFllower(Canvas canvas, List<Fllower> fllowers) {
        for (Fllower fllower : fllowers) {
```

```
            float[] pos = new float[2];
            canvas.drawPath(fllower.getPath(), mPaint);
            pathMeasure.setPath(fllower.getPath(), false);
            pathMeasure.getPosTan(height * fllower.getValue(), pos, null);
            canvas.drawBitmap(mBitmap, pos[0], pos[1] - top, null);
        }
    }
```

首先，遍历一个Fllower集合，然后把每个Fllower所属的Path画出来，就是上面蓝色的曲线，然后很眼熟了吧，给PathMeasure设置Path对象，然后呢，就是重点啦！height是屏幕的高度，fllower.getValue()也是一个百分比，从0-1，和前面的Animator作用相同，这句代码就是说，我要距离为height * fllower.getValue()处的点的坐标，给我放在pos里面！

好了，点的坐标都有了，剩下的还需要说么...

不行了，再不回家，就真回不去了，拜拜，同学们

更多参考资料

- Path特效之PathMeasure打造万能路径动效
- android 路径动画制作