# 自定义View：仿QQ拖拽效果

史大拿 郭霖 2022-08-26 08:00 发表于江苏



## / 今日科技快讯 /

近日，支付宝"借呗"板块新增信用卡取现入口一事引发热议。通过支付宝与银行的合作，用户可以直接在支付宝页面进行名下信用卡的取现操作。信息显示，当前已经有宁波银行、光大银行以及平安银行等3家信用卡支持取现，但仅有内测用户可以查看对应板块入口。

## / 作者简介 /

明天就是愉快的周六啦，提前祝大家周末愉快！

本篇文章来自自史大拿的投稿，文章主要分享了水滴式的拖动效果，相信会对大家有所帮助！同时也感谢作者贡献的精彩文章！

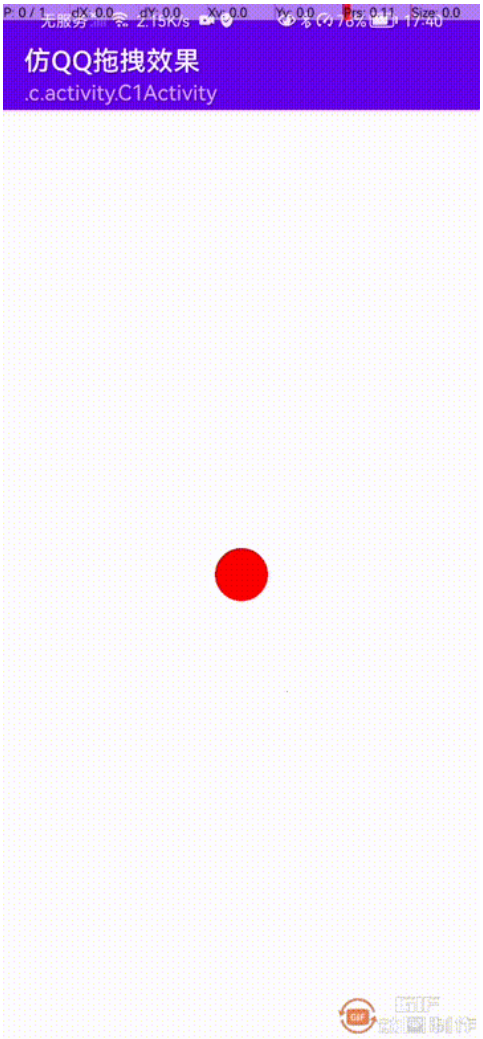
史大拿的博客地址：

> https://blog.csdn.net/weixin_44819566?type=blog

## / 前言 /

- **android studio**: 4.1.3
- **kotlin version**:1.5.0
- **gradle**: gradle-6.5-bin.zip

废话不多说,先来看今天要完成的效果:


仿QQ拖拽效果
.c.activity.C1Activity


图二是在图一的基础上改的,可以通过一行代码,让所有控件都能实现拖拽效果!

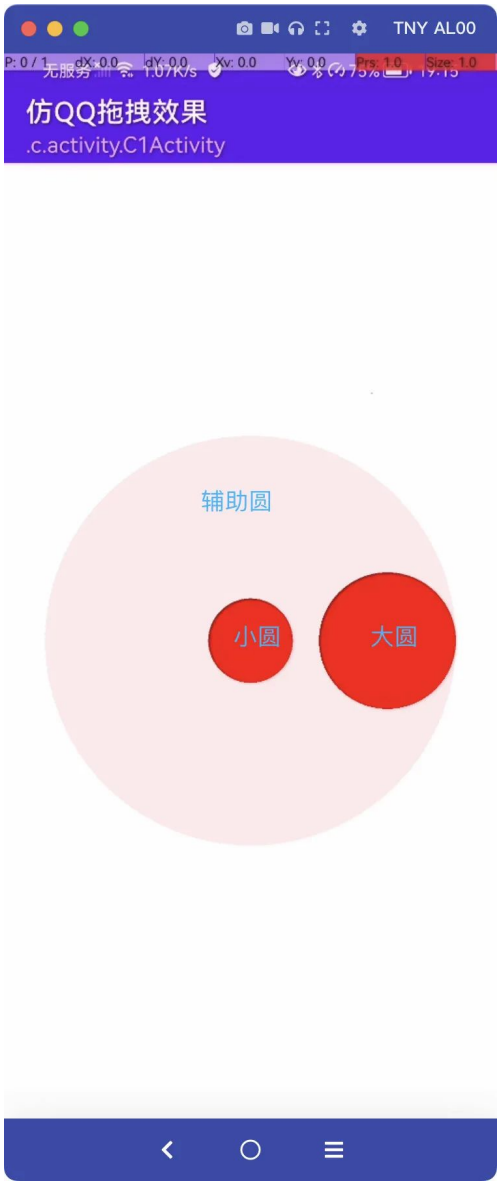
所以先来编写效果一的代码~

## / 基础绘制 /

首先编写一下基础代码:

```
class TempView @JvmOverloads constructor(
    context: Context, attrs: AttributeSet? = null, defStyleAttr: Int = 0,
) : View(context, attrs, defStyleAttr) {

    companion object {
        // 大圆半径
        private val BIG_RADIUS = 50.dp

        // 小圆半径
        private val SMALL_RADIUS = BIG_RADIUS * 0.618f
```

```
        // 最大范围(半径)，超出这个范围大圆不显示
        private val MAX_RADIUS = 150.dp
    }

    private val paint = Paint().apply {
        color = Color.RED
    }

    // 大圆初始位置
    private val bigPointF by lazy { PointF(width / 2f + 300, height / 2f) }

    // 小圆初始位置
    private val smallPointF by lazy { PointF(width / 2f, height / 2f) }

    override fun onDraw(canvas: Canvas) {
        super.onDraw(canvas)

        paint.color = Color.RED
        // 绘制大圆
        canvas.drawCircle(bigPointF.x, bigPointF.y, BIG_RADIUS, paint)

        // 绘制小圆
        canvas.drawCircle(smallPointF.x, smallPointF.y, SMALL_RADIUS, paint)

        // 绘制辅助圆
        paint.color = Color.argb(20, 255, 0, 0)
        canvas.drawCircle(smallPointF.x, smallPointF.y, MAX_RADIUS, paint)
    }
}
```




这段代码很简单,都是一些基础api的调用,辅助圆的作用:

- 当大圆**超出**辅助圆范围的时候,大圆得“爆炸”,
- 如果大圆**未超出**辅助圆内的话,大圆得回弹回去~

主要就是起到这样的作用.

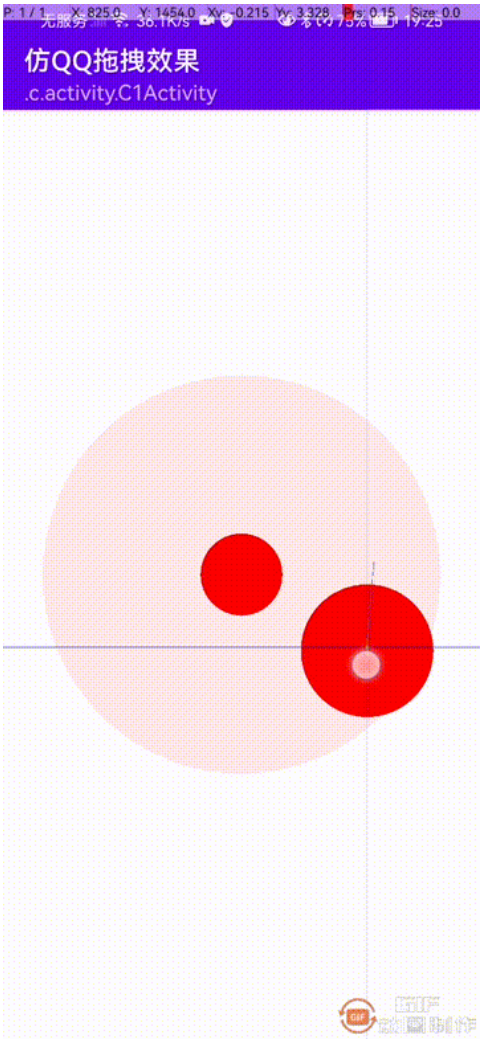
### 大圆动起来

```
override fun onTouchEvent(event: MotionEvent): Boolean {
  when (event.action) {
    MotionEvent.ACTION_DOWN -> {

    }
```

```
            MotionEvent.ACTION_MOVE -> {
                bigPointF.x = event.x
                bigPointF.y = event.y
            }
            MotionEvent.ACTION_UP -> {

            }
        }
        invalidate()
        return true // 消费事件
    }
```

大圆动起来很简单,只需要在ACTION_MOVE中一直刷新移动位置即可

**辅助图1.1:**




我们想要的效果是手指按下之后,大圆跟着移动,

在**辅助图1.1**后半段可以看出这里有一个小问题，手指按什么位置小球就移动到什么位置,不是我们想要的效果

那么我们知道所有的事件都是在DOWN中分发出来的，

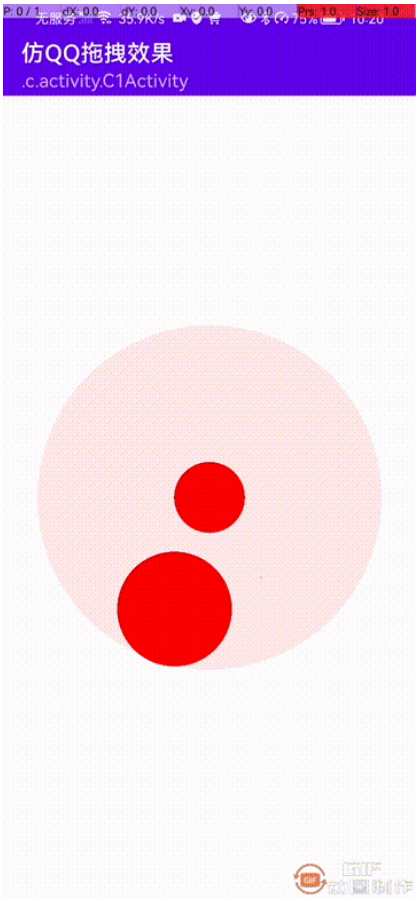
所以只需要在DOWN事件中判断当前是否点击到大圆即可，

```
// 标记是否选中了大圆
var isMove = false

@SuppressLint("ClickableViewAccessibility")
override fun onTouchEvent(event: MotionEvent): Boolean {
   when (event.action) {
      MotionEvent.ACTION_DOWN -> {
       // 判断当前点击区域是否在大圆范围内
        isMove = bigPointF.contains(PointF(event.x, event.y), BIG_RADIUS)
      }
      MotionEvent.ACTION_MOVE -> {
        if (isMove) {
           bigPointF.x = event.x
           bigPointF.y = event.y
        }
      }
   }
   invalidate()
   return true // 消费事件
}
```

contains是自己写的一个扩展函数:

```
// 判断一个点是否在另一个点内
fun PointF.contains(b: PointF, bPadding: Float = 0f): Boolean {
   val isX = this.x <= b.x + bPadding && this.x >= b.x - bPadding

   val isY = this.y <= b.y + bPadding && this.y >= b.y - bPadding
   return isX && isY
}
```

**辅助图1.2:**



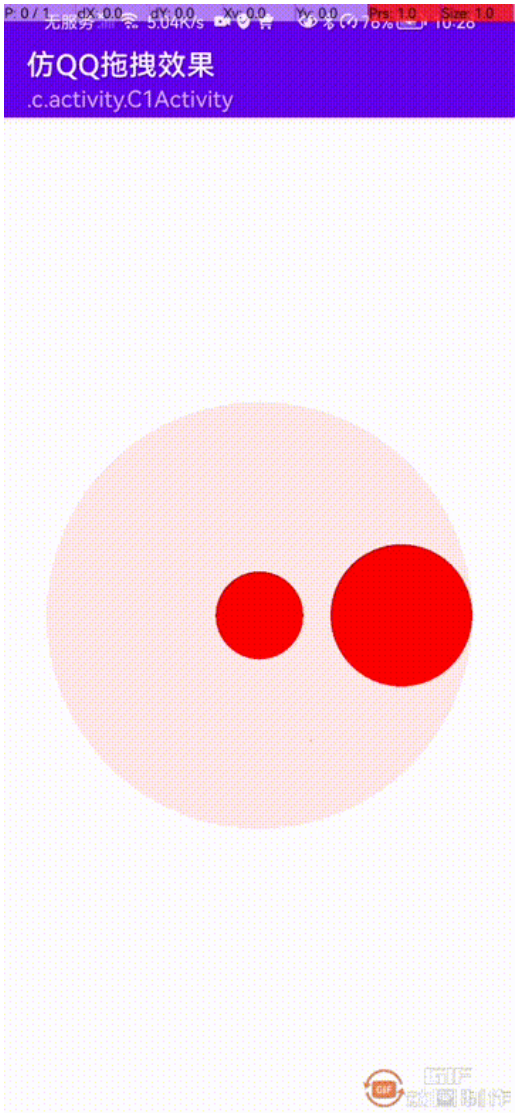

**大圆超出辅助圆范围就消失**

有了**PointF.contains()** 这个扩展,任务就变得轻松起来了

只需要在绘制的时候判断一下当前位置即可

```
override fun onDraw(canvas: Canvas) {
   super.onDraw(canvas)
    // 大圆位置是否在辅助圆内
    if(bigPointF.contains(smallPointF, MAX_RADIUS)){
      // 绘制大圆
      canvas.drawCircle(bigPointF.x, bigPointF.y, BIG_RADIUS, paint)
    }
     // 绘制小圆
    ...

   // 绘制辅助圆
   ...
}
```

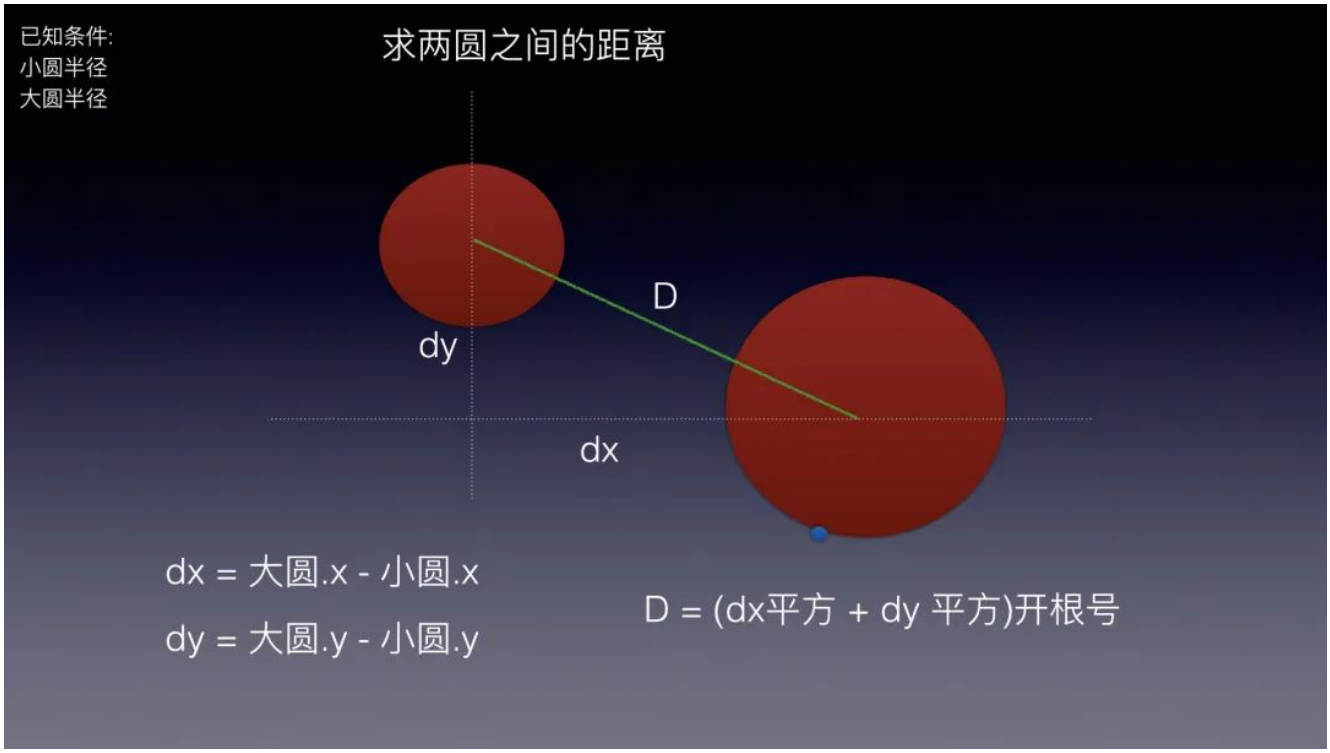
**辅助图1.3:**




### 大圆越往外,小球越小

要想求出大圆是否越往外,那么就得先计算出当前大圆与小圆的距离

**辅助图1.4:**




- dx = 大圆.x - 小圆.x
- dy = 大圆.y - 小圆.y

通过勾股定理就可以计算出他们之间的距离

```
// 小圆与大圆之间的距离
private fun distance(): Float {
  val current = bigPointF - smallPointF
  return sqrt(current.x.toDouble().pow(2.0) + (current.y.toDouble().pow(2.0))).toFloat()
}
```

bigPointF - smallPointF 采用的是ktx中自带的运算符重载函数

知道大圆和小圆的距离之后,就可以计算出比例

比例 = 距离 / 总长度

```
// 大圆与小圆之间的距离
val d = distance()

// 总长度
var ratio = d / MAX_RADIUS
// 如果当前比例 > 0.618 那么就让=0.618
if (ratio > 0.618) {
```

```
    ratio = 0.618f
}
```

为什么要选0.618,

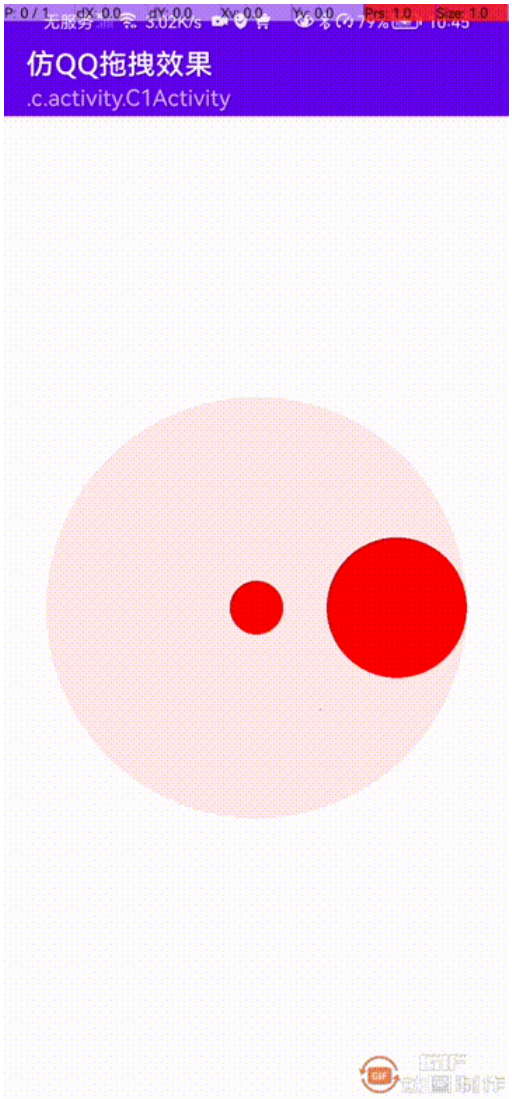
0.618是黄金比例分割点,听说选了0.618绘制出来的东西会很协调?

我一个糙人也看不出来美不美, 可能是看着更专业一点吧.

完整绘制小圆代码:

```
//小圆半径
private val SMALL_RADIUS = BIG_RADIUS

override fun onDraw(canvas: Canvas) {
    super.onDraw(canvas)

     // 绘制大圆
      ...

    // 两圆之间的距离
    val d = distance()
    var ratio = d / MAX_RADIUS
    if (ratio > 0.618) {
        ratio = 0.618f
    }
    // 小圆半径
    val smallRadius = SMALL_RADIUS - SMALL_RADIUS * ratio
    // 绘制小圆
    canvas.drawCircle(smallPointF.x, smallPointF.y, smallRadius, paint)


    // 绘制辅助圆
   ...
}
```

**辅助图1.5:**




### 绘制贝塞尔曲线

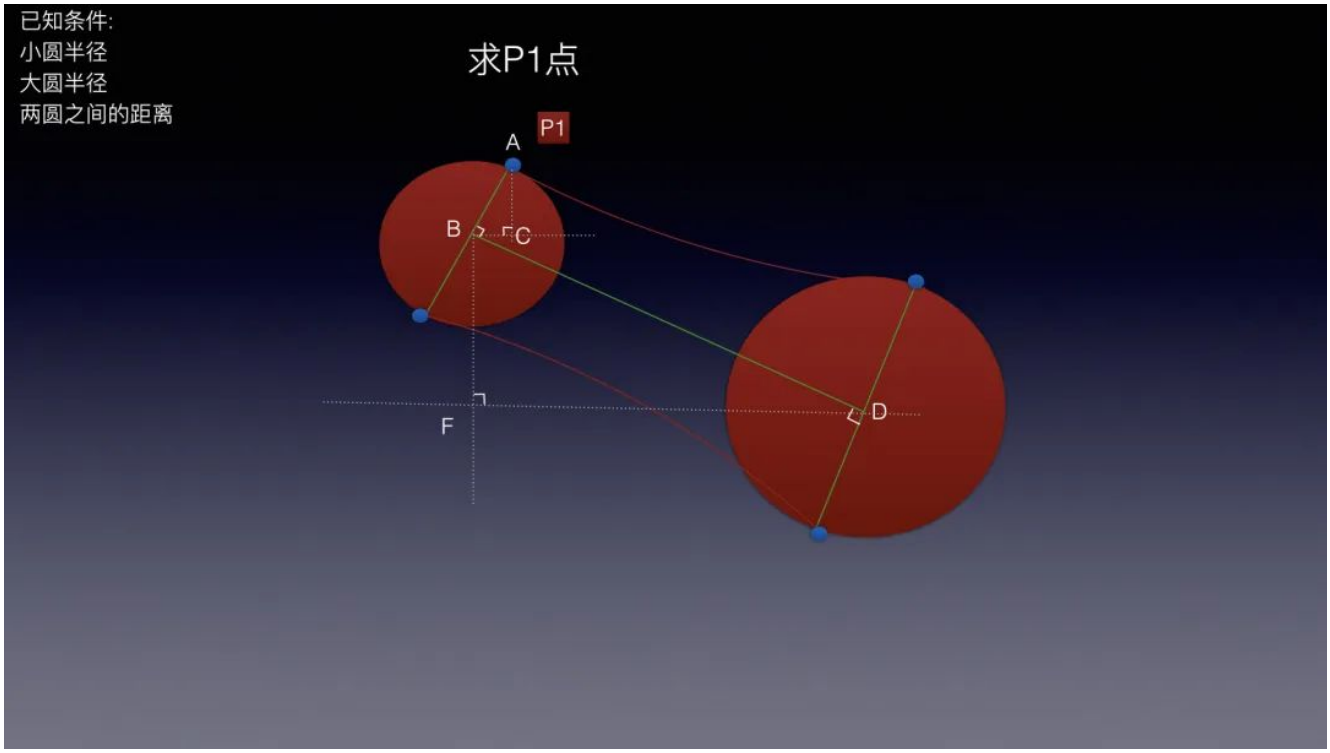
接下来只需要求出这4个点连接起来，看起来就像是把他们连接起来了

然后在找到一个控制点，通过贝塞尔曲线让他稍微弯曲即可

**辅助图1.6:**

- **P1**

**辅助图1.7:**




最终就是算出角A的坐标即可

目前已知

- 角A.x = 小圆.x + BC;
- 角A.y = 小圆.y - AC ;

*Tips: 因为角A的坐标在小圆中心点上面, 在android坐标系中 角A.y = 小圆.y - AC ;*

- 角C = 90度;
- 角ABD = 90度

角ABC + 角BAC = 90度; 角ABC +角CBD = 90度;

所以角BAC = 角CBD

BC 平行于 FD,那么**角BDF = 角CBD = 角A**

最终**只要求出角BDF就算出了角A**

假设现在知道角A, AB的长度 = 小圆的半径

就可以算出:

- BC = AB * sin(角A)
- AC = AB * cos(角A)

现在已知BF 和 FD的距离

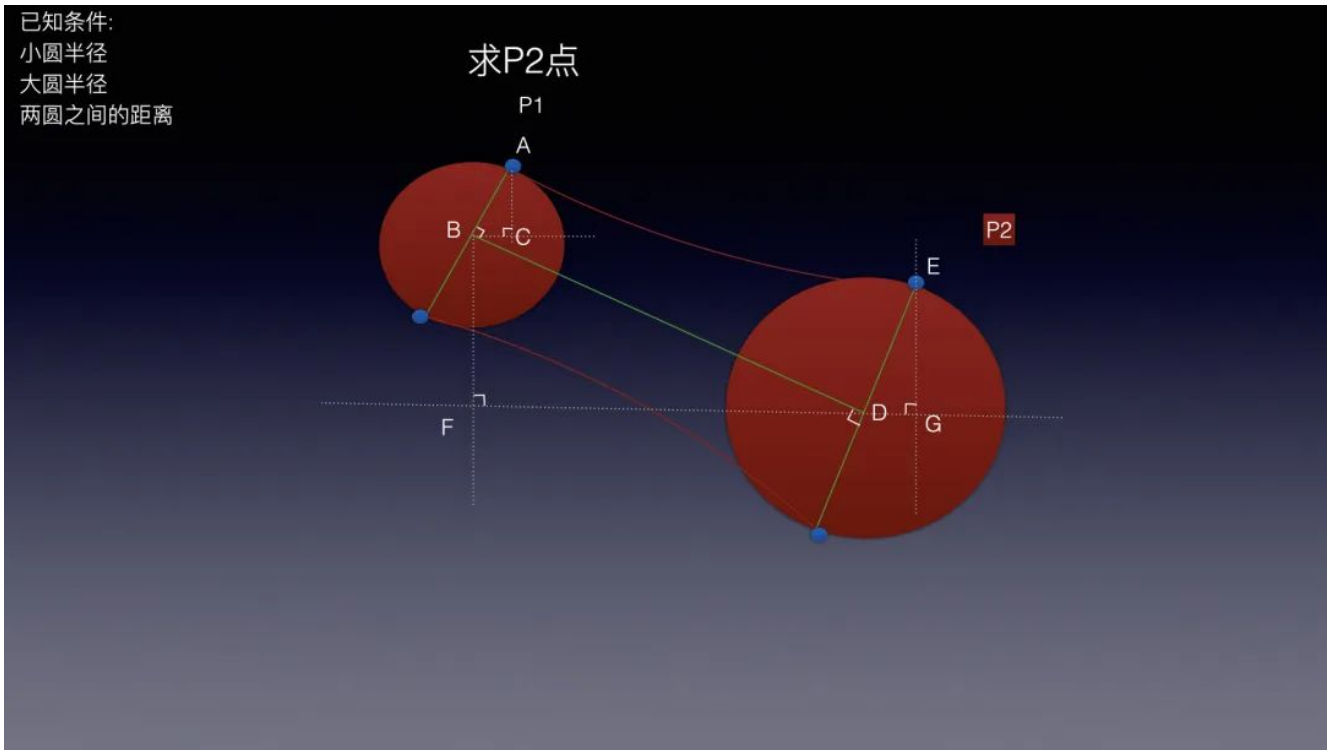
角BDF = arctan(BF / FD)

那么现在就计算出了角A的角度

- p1X = 小圆.x + 小圆半径 * sin(角A)
- p1Y = 小圆.y - 小圆半径 * cos(角A)

- **P2**

**辅助图1.8:**




现在要求出P2的位置,也就是角E的位置

- 角E.x = 大圆.x + DG
- 角E.y = 大圆.y + EG

角BDE = 90度;

角BDF + 角EDG = 90度

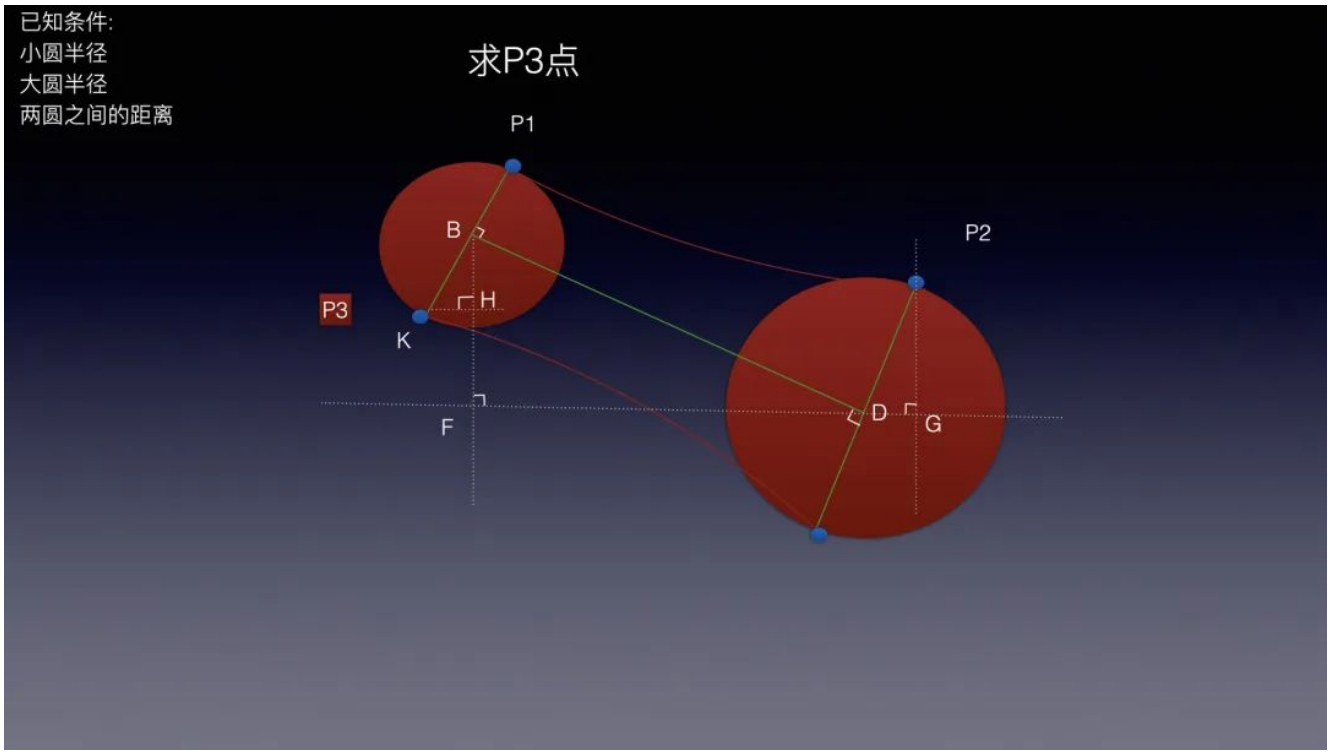
那么角E = 角BDF

**P1**刚刚计算了角BDF,还是热的.

- P2.x =大圆.x + DE * sin(角E)
- P2.y = 大圆.y - DE * cos(角E)

- **P3**

**辅助图1.9:**




P3就是角K的位置

- 角K.x = 小圆.x - KH
- 角K.y = 小圆.y - BH

角KBH + 角HBD = 90度

角BDF + 角HBD = 90度

所以角KBH + 角BDF

KH = BK * sin(角KBH)

BK = BK * cos(角KBH)

- P3.x = 小圆.x - KH

- P3.y = 小圆.y - BH

- **P4**

**辅助图1.10:**

- 角A.x = 大圆.x - CD
- 角A.y = 大圆.y + AC

角A + 角ADC = 90度

角BDF + 角ADC = 90度

所以角A = 角BDF

CD = AD * sin(角A)

AC = AD * cos(角A)

- P4.x = 大圆.x - CD

- p4.y = 大圆.y - AC

- **控制点**

控制点就选大圆与小圆的中点即可

控制点.x = (大圆.x - 小圆.x) / 2 + 小圆.x

控制点.y = (大圆.y - 小圆.y) / 2 + 小圆.y

来看看完整代码:

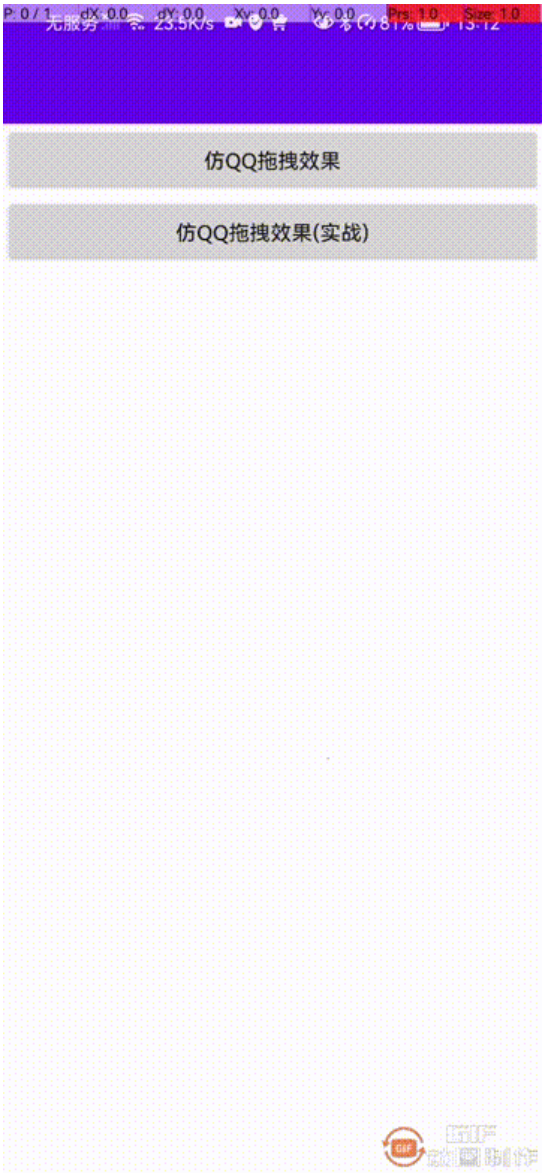
```
/*
* 作者:史大拿
* @param smallRadius: 小圆半径
* @param bigRadius: 大圆半径
*/
 private fun drawBezier(canvas: Canvas, smallRadius: Float, bigRadius: Float) {
    val current = bigPointF - smallPointF

    val BF = current.y.toDouble()
    val FD = current.x.toDouble()
    //
    val BDF = atan(BF / FD)

    val p1X = smallPointF.x + smallRadius * sin(BDF)
    val p1Y = smallPointF.y - smallRadius * cos(BDF)

    val p2X = bigPointF.x + bigRadius * sin(BDF)
    val p2Y = bigPointF.y - bigRadius * cos(BDF)

    val p3X = smallPointF.x - smallRadius * sin(BDF)
    val p3Y = smallPointF.y + smallRadius * cos(BDF)

    val p4X = bigPointF.x - bigRadius * sin(BDF)
    val p4Y = bigPointF.y + bigRadius * cos(BDF)

    // 控制点
    val controlPointX = current.x / 2 + smallPointF.x
    val controlPointY = current.y / 2 + smallPointF.y

    val path = Path()
    path.moveTo(p1X.toFloat(), p1Y.toFloat()) // 移动到p1位置
    path.quadTo(controlPointX, controlPointY, p2X.toFloat(), p2Y.toFloat()) // 绘制贝塞尔
```

```
    path.lineTo(p4X.toFloat(), p4Y.toFloat()) // 连接到p4
    path.quadTo(controlPointX, controlPointY, p3X.toFloat(), p3Y.toFloat()) // 绘制贝塞尔
    path.close() // 连接到p1
    canvas.drawPath(path, paint)
}
```

调用:

```
override fun onDraw(canvas: Canvas) {
    super.onDraw(canvas)

    paint.color = Color.RED

    // 两圆之间的距离
    val d = distance()
    var ratio = d / MAX_RADIUS
    if (ratio > 0.618) {
        ratio = 0.618f
    }
    // 小圆半径
    val smallRadius = SMALL_RADIUS - SMALL_RADIUS * ratio
    // 绘制小圆
    canvas.drawCircle(smallPointF.x, smallPointF.y, smallRadius, paint)

    // 大圆位置是否在辅助圆内
    if (bigPointF.contains(smallPointF, MAX_RADIUS)) {
        // 绘制大圆
        canvas.drawCircle(bigPointF.x, bigPointF.y, BIG_RADIUS, paint)

        // 绘制贝塞尔
        drawBezier(canvas,smallRadius, BIG_RADIUS)
    }

    // 绘制辅助圆
    ...
}
```

**辅助图1.11:**

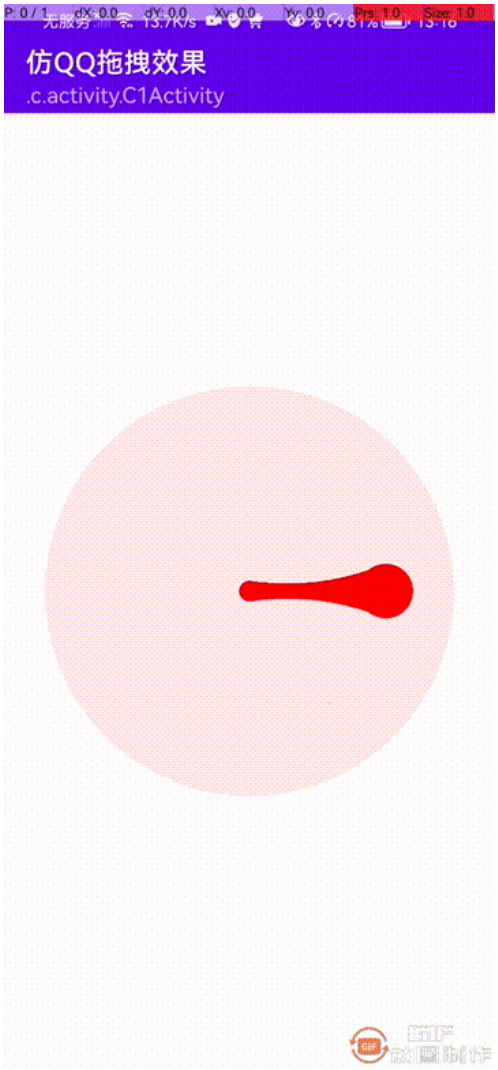



可以看出,基本效果已经达到了,但是这个大圆看着很大,总感觉有地方不协调

那是因为这些参数都是我自己随便写的,到时候这些参数UI都会给你,肯定没有这么随意...

可以先吧大圆半径缩小一点再看看效果如何

**辅助图1.12:**




看着效果其实还可以.

## / 拖动回弹 /

拖动回弹是指当拖动大圆时候,没有超出辅助圆的范围, 此时大圆还在辅助圆范围内,

那么就需要将大圆回弹到小圆位置上.

那么肯定是松手(ACTION_UP)事件的时候来处理:

```
private fun bigAnimator(): ValueAnimator {
    return ObjectAnimator.ofObject(this, "bigPointF", PointFEvaluator(),
        PointF(width / 2f, height / 2f)).apply {
        duration = 400
        interpolator = OvershootInterpolator(3f) // 设置回弹迭代器
    }
}
```

常见插值器:

- AccelerateDecelerateInterpolator 动画从开始到结束，变化率是先加速后减速的过程。
- AccelerateInterpolator 动画从开始到结束，变化率是一个加速的过程。
- AnticipateInterpolator 开始的时候向后，然后向前甩
- AnticipateOvershootInterpolator 开始的时候向后，然后向前甩一定值后返回最后的值
- BounceInterpolator 动画结束的时候弹起
- CycleInterpolator 动画从开始到结束，变化率是循环给定次数的正弦曲线。
- DecelerateInterpolator 动画从开始到结束，变化率是一个减速的过程。
- LinearInterpolator 以常量速率改变
- OvershootInterpolator 结束时候向反方向甩某段距离

插值器参考链接(https://cloud.tencent.com/developer/article/1488956)

最开始初始化大圆位置为:

```
private val bigPointF by lazy { PointF(width / 2f + 300, height / 2f) }
```

此时通过动画来改变bigPointF肯定是不可取的,因为他是懒加载

所以要修改初始化代码为:

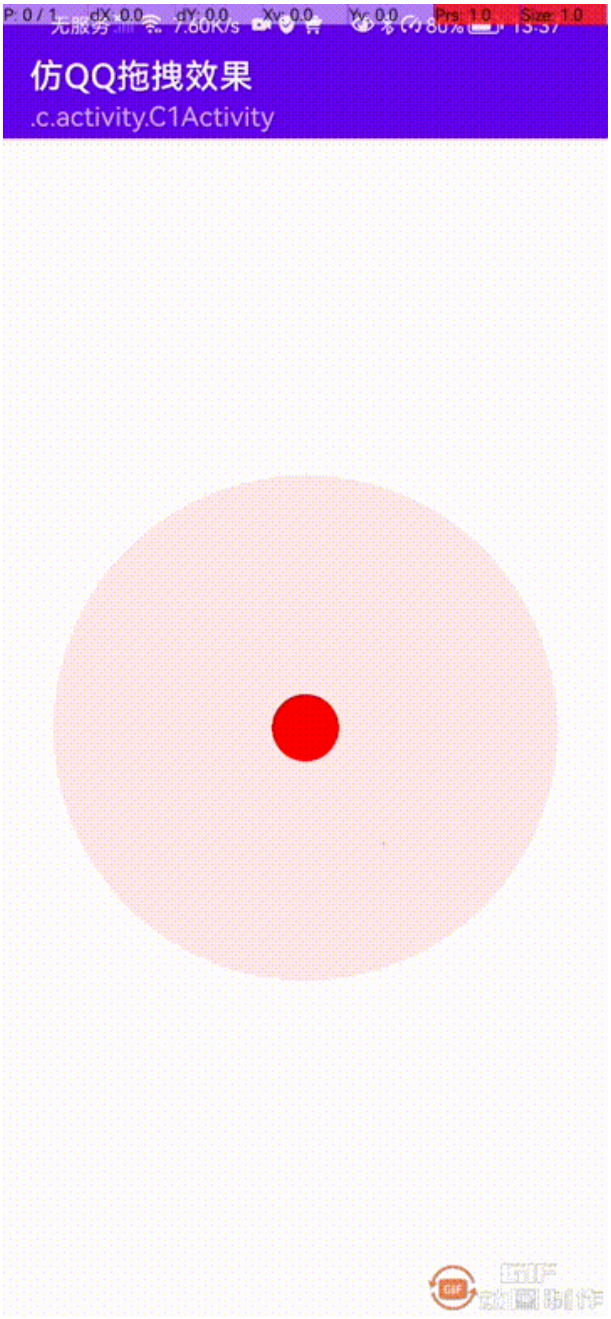
```
var bigPointF = PointF(0f, 0f)
 set(value) {
  field = value
  invalidate()
 }

override fun onSizeChanged(w: Int, h: Int, oldw: Int, oldh: Int) {
   super.onSizeChanged(w, h, oldw, oldh
   bigPointF.x = width / 2f
   bigPointF.y = height / 2f
}
```

如果对为什么要在onSizeChanged中调用不明白的建议看一下View生命周期

调用：

```
override fun onTouchEvent(event: MotionEvent): Boolean {
    when (event.action) {
        ....
        MotionEvent.ACTION_UP -> {
            // 大圆是否在辅助圆范围内
            if (bigPointF.contains(smallPointF, MAX_RADIUS)) {
                // 回弹
                bigAnimator().start()
            } else {
                // 爆炸
            }
        }
    }
    invalidate()
    return true // 消费事件
}
```

**辅助图1.13：**




最后当大圆拖动到辅助圆外的时候,在UP位置绘制爆炸效果,

并且当爆炸效果结束时候,吧大圆x,y坐标回到小圆坐标即可!

## / 爆炸效果 /

爆炸效果其实就是20张图片一直在切换,达到一帧一帧的效果即可

```
private val explodeImages by lazy {
    val list = arrayListOf<Bitmap>()
    // BIG_RADIUS = 大圆半径
    val width = BIG_RADIUS * 2 * 2
```

```
    list.add(getBitMap(R.mipmap.explode_0, width.toInt()))
    list.add(getBitMap(R.mipmap.explode_1, width.toInt()))
    list.add(getBitMap(R.mipmap.explode_2, width.toInt()))
    list.add(getBitMap(R.mipmap.explode_3, width.toInt()))
    list.add(getBitMap(R.mipmap.explode_4, width.toInt()))
    list.add(getBitMap(R.mipmap.explode_5, width.toInt()))
    list.add(getBitMap(R.mipmap.explode_5, width.toInt()))
    list.add(getBitMap(R.mipmap.explode_6, width.toInt()))
    list.add(getBitMap(R.mipmap.explode_7, width.toInt()))
    list.add(getBitMap(R.mipmap.explode_8, width.toInt()))
    list.add(getBitMap(R.mipmap.explode_9, width.toInt()))
    list.add(getBitMap(R.mipmap.explode_10, width.toInt()))
    list.add(getBitMap(R.mipmap.explode_11, width.toInt()))
    list.add(getBitMap(R.mipmap.explode_12, width.toInt()))
    list.add(getBitMap(R.mipmap.explode_13, width.toInt()))
    list.add(getBitMap(R.mipmap.explode_14, width.toInt()))
    list.add(getBitMap(R.mipmap.explode_15, width.toInt()))
    list.add(getBitMap(R.mipmap.explode_16, width.toInt()))
    list.add(getBitMap(R.mipmap.explode_17, width.toInt()))
    list.add(getBitMap(R.mipmap.explode_18, width.toInt()))
    list.add(getBitMap(R.mipmap.explode_19, width.toInt()))
    list
}

// 爆炸下标
var explodeIndex = -1
    set(value) {
        field = value
        invalidate()
    }

// 属性动画修改爆炸下标,最后一帧的时候回到 -1
private val explodeAnimator by lazy {
        ObjectAnimator.ofInt(this, "explodeIndex", 19, -1).apply {
            duration = 1000
        }
    }
```

调用：

```
override fun onTouchEvent(event: MotionEvent): Boolean {
    when (event.action) {
            ...
        MotionEvent.ACTION_UP -> {
            // 大圆是否在辅助圆范围内
            if (bigPointF.contains(smallPointF, MAX_RADIUS)) {
                // 回弹
                 ....
            } else {
```

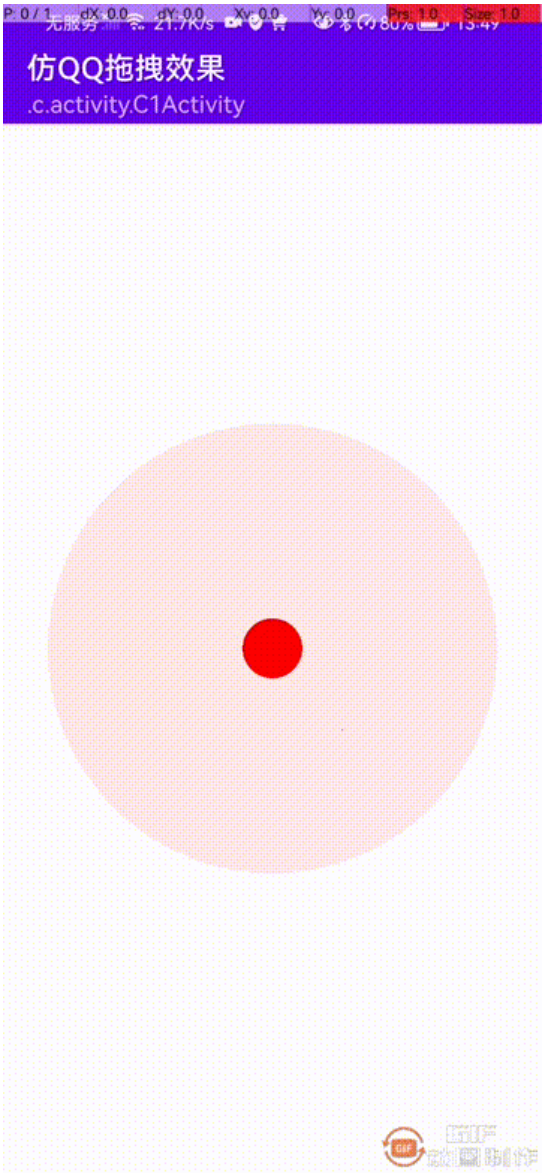
```
            // 绘制爆炸效果
            explodeAnimator.start()
            // 爆炸效果结束后，将图片移动到原始位置
            explodeAnimator.doOnEnd {
               bigPointF.x = width / 2f
               bigPointF.y = height / 2f
            }
         }
      }
   }
   invalidate()
   return true // 消费事件
}
```

绘制BitMap:

```
override fun onDraw(canvas: Canvas) {
   super.onDraw(canvas)
   // 绘制小圆
   ...

   // 大圆位置是否在辅助圆内
   if (bigPointF.contains(smallPointF, MAX_RADIUS)) {
      // 绘制大圆
      ....

      // 绘制贝塞尔
      ...
   }

   // 绘制爆炸效果
   if (explodeIndex != -1) {
      // 圆和bitmap坐标系不同
      // 圆的坐标系是中心点
      // bitmap的坐标系是左上角
      canvas.drawBitmap(explodeImages[explodeIndex],
         bigPointF.x - BIG_RADIUS * 2,
         bigPointF.y - BIG_RADIUS * 2,
         paint)
   }

   // 绘制辅助圆
    ....
}
override fun onDraw(canvas: Canvas) {
   super.onDraw(canvas)
   // 绘制小圆
   ...
```

```
    // 大圆位置是否在辅助圆内
    if (bigPointF.contains(smallPointF, MAX_RADIUS)) {
      // 绘制大圆
      ....

      // 绘制贝塞尔
      ...
    }

    // 绘制爆炸效果
    if (explodeIndex != -1) {
      // 圆和bitmap坐标系不同
      // 圆的坐标系是中心点
      // bitmap的坐标系是左上角
      canvas.drawBitmap(explodeImages[explodeIndex],
        bigPointF.x - BIG_RADIUS * 2,
        bigPointF.y - BIG_RADIUS * 2,
        paint)
    }

    // 绘制辅助圆
     ....
}
```

### 辅助图1.14:



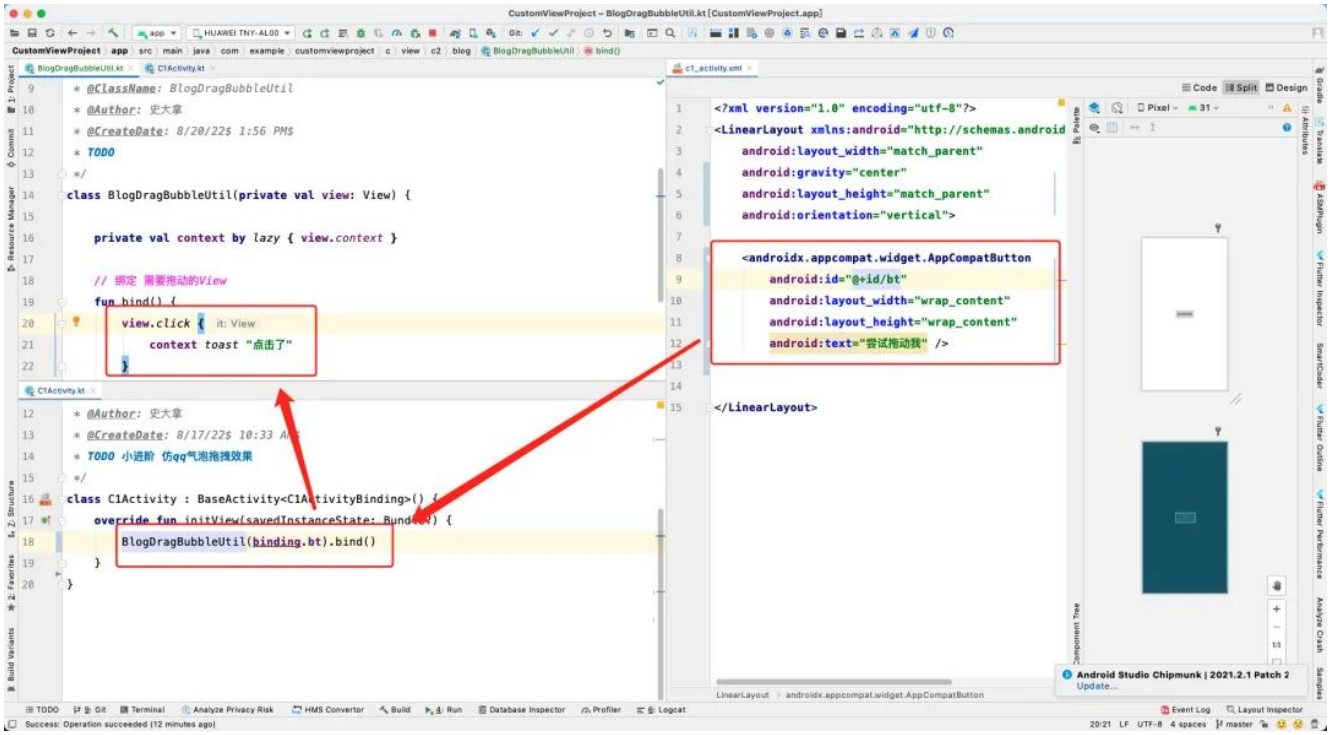

虽然爆炸效果绘制出来了,但是看着爆炸时候稍微稍微有一点抖动,

这是因为爆炸效果的这20张图片是我在网上找的,一张一张切出来的... 手艺不好,可能有一点歪歪扭扭,但是不打紧,到时候UI都会给你的@.@

到此时,效果一就完成了...

## / 效果二 /

赛前准备:




要想拖动View,那就必须有一个View,那么就以这个Button来演示吧~

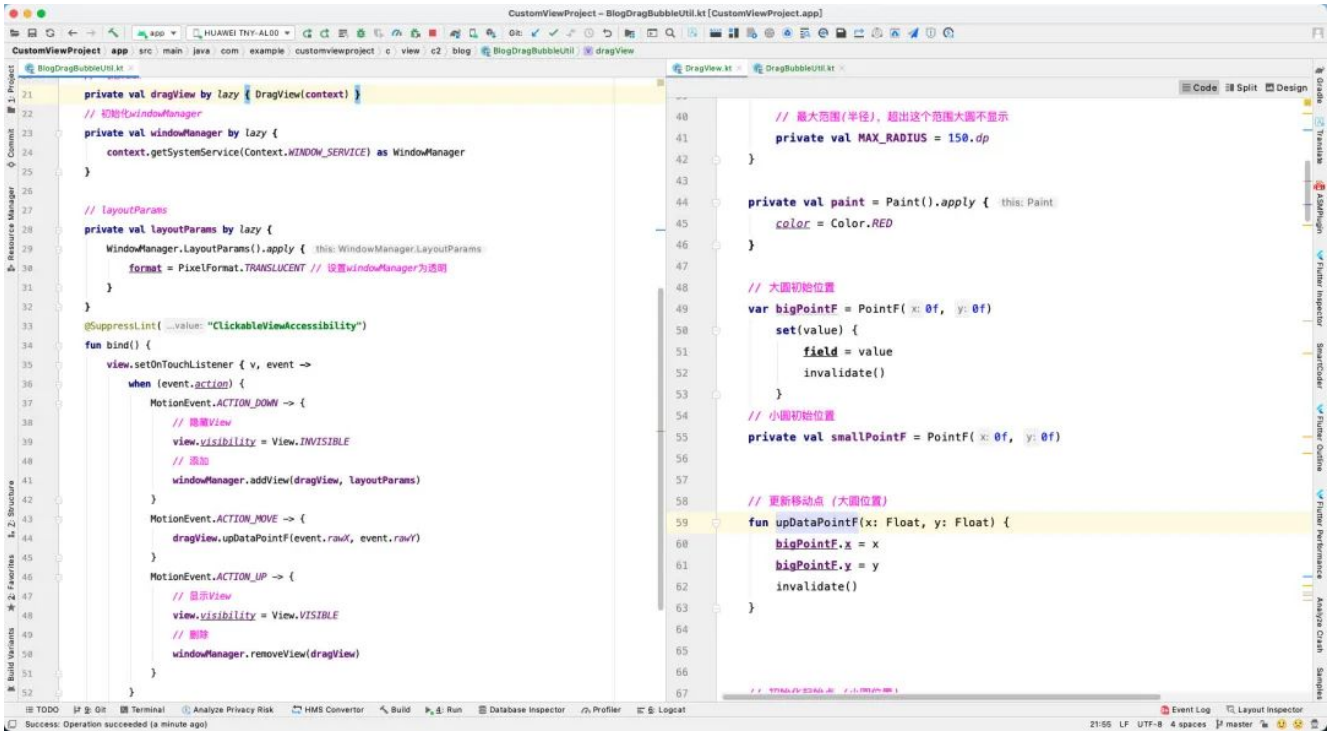

思路分析:

1. 通过setOnTouchListener{} 可以实现对View的触摸事件监听
2. 在ACTION_DOWN事件时候,将当前View隐藏,通过WindowManager添加一个拖拽的气泡View((就是上面写好的), 并且给气泡View初始化好位置
3. 在ACTION_MOVE 事件中不断的更新大圆的位置
4. 在ACTION_UP事件的时候,判断是否在辅助圆内,然后进行回弹或者爆炸. 并且将拖拽气泡从WindowManager总删除掉

需要注意的是,既然通过setOnTouchListener{} 监听了移动位置,那么在拖拽View中onTouchEvent()中的代码全都要删掉

这只是总体思路,还有很多细节,那就慢慢分析吧~

## 将dragView添加WindowManager上




可以看出,现在有2个问题

- 初始化位置不对
- 当拖动的时候,状态栏变成了黑色

### 初始化位置不对

初始化位置不对,需要有2个初始点

- 小圆初始点: 小圆初始点既是当前view的中心点
- 大圆初始点: 大圆初始点即是当前按下的位置

当前View的中心点需要获取当前window的绝对坐标位置:

```
// location[0] = x;
// location[1] = y;
```

```
val location = IntArray(2)
view.getLocationInWindow(location) // 获取当前窗口的绝对坐标
```

大圆位置为当前点击屏幕的绝对位置:

即为 event.rawX; even.rawY

调用:

```
#BlogDragBubbleUtil.kt

when (event.action) {
 MotionEvent.ACTION_DOWN -> {
      val location = IntArray(2)
   view.getLocationInWindow(location) // 获取当前窗口的绝对坐标
   dragView.initPointF(
    location[0].toFloat() + view.width / 2,
    location[1].toFloat()+ view.height / 2 ,
    event.rawX,
    event.rawY
  )
 }
 ....
}
```

可以看出,基本已经没问题了,但是点击的,明显有一点偏下,这是因为绝对位置不包含状体栏的高度

所以需要在减去状态栏的高度即可

```
// 获取状态栏高度
fun Context.statusBarHeight() = let {
   var height = 0
   val resourceId: Int = resources
      .getIdentifier("status_bar_height", "dimen", "android")
   if (resourceId > 0) {
      height = resources.getDimensionPixelSize(resourceId)
   }
   height
}
```

最终代码为:

```
// 屏幕状态栏高度
private val statusBarHeight by lazy {
 context.statusBarHeight()
}
```

```
MotionEvent.ACTION_DOWN -> {
 dragView.initPointF(
   location[0].toFloat() + view.width / 2,
   location[1].toFloat() + view.height / 2 - statusBarHeight,
   event.rawX,
   event.rawY - statusBarHeight
 )
}
MotionEvent.ACTION_MOVE -> {
  dragView.upDataPointF(event.rawX, event.rawY - statusBarHeight)
}
```

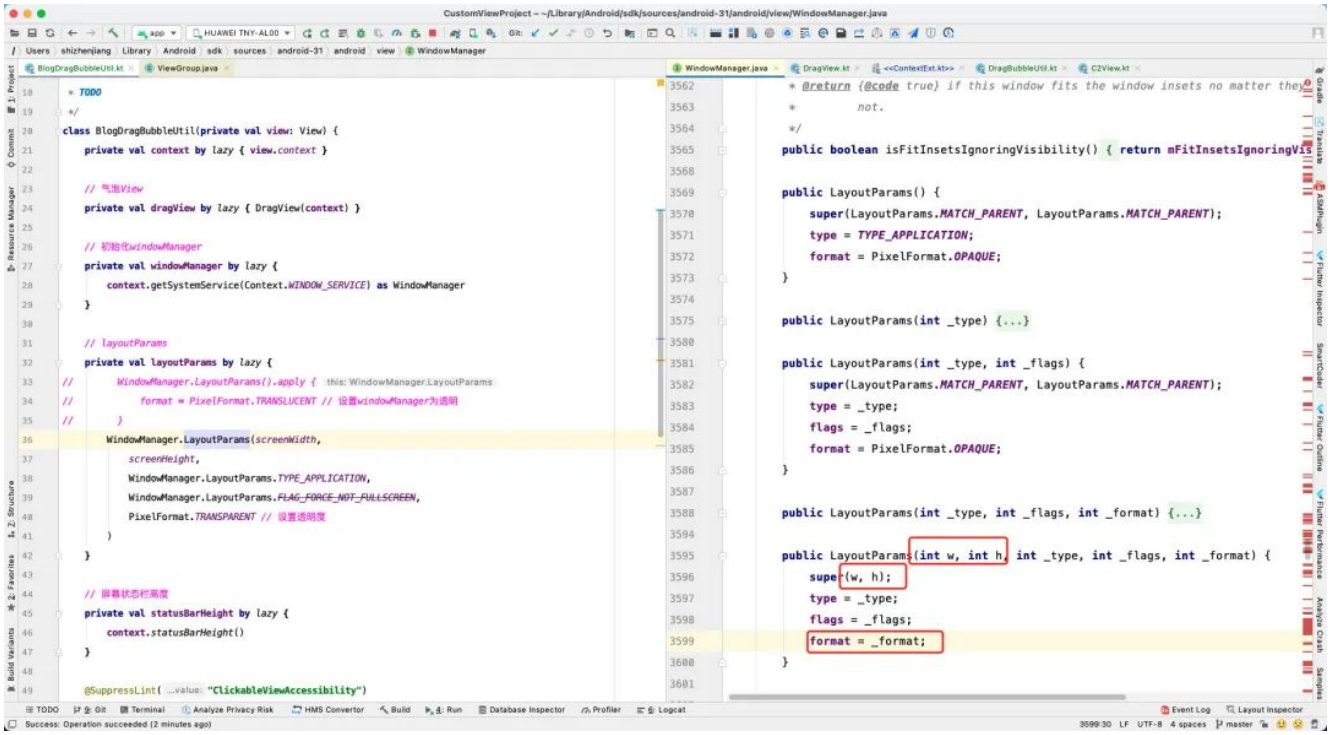
### 当拖动的时候,状态栏变成了黑色

状态栏变成黑色,说明是LayoutParams 完全占满了整个屏幕

那么只需要手动给他不包含状态栏的高度即可

```
private val layoutParams by lazy {
//     WindowManager.LayoutParams().apply {
//        format = PixelFormat.TRANSLUCENT // 设置windowManager为透明
//     }
     WindowManager.LayoutParams(screenWidth,
        screenHeight,
        WindowManager.LayoutParams.TYPE_APPLICATION,
        WindowManager.LayoutParams.FLAG_FORCE_NOT_FULLSCREEN,
        PixelFormat.TRANSPARENT // 设置透明度
     )
  }
```

来看看WindowManager.LayoutParams的源码




这里我调用的是5个参数的,我也没研究过这5个参数是干嘛用的

我只认识三个

- 宽
- 高
- 透明度

type 和 flags我都是设置的默认值.

无论如何这么写是可行的,当前效果

现在流程已经走了50%

### 复制drawView中BitMap,并且绘制

这个标题我有必要解释一下,每一个控件内其实都是bitmap绘制的

我想要的效果是当拖动的时候,拖动的是控件,而不是变成一个红色的小球

所以在拖动过程中,我们要复制出drawView中bitmap图片,然后在重新绘制一下

肉眼看就已经达到了效果,但是对于代码来说,本体已经隐藏了,只是留下了一个复制品来展示而已

```
// 从View中获取bitMap
fun View.getBackgroundBitMap(): Bitmap = let {
```

```
    this.buildDrawingCache()
    this.drawingCache
}
```

在DOWN中设置bitMap图片

在UP的时候清空图片

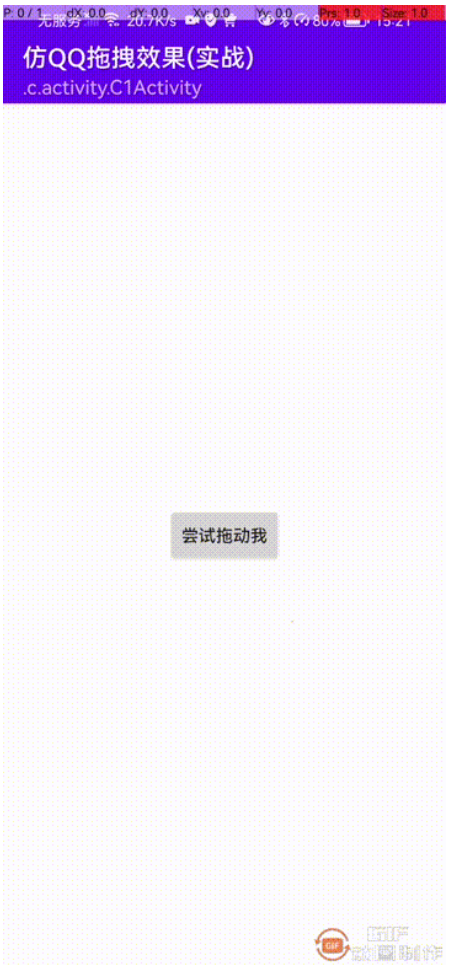
```
#BlogDragBubbleUtil.kt
// view的图片
private val bitMap by lazy { view.getBackgroundBitMap() }

fun bind() {
     view.setOnTouchListener { v, event ->
       when (event.action) {
          MotionEvent.ACTION_DOWN -> {
           // 初始化位置
            dragView.initPointF(..)


            // 设置BitMap图片
            dragView.upDataBitMap(bitMap, bitMap.width.toFloat())
          }
          MotionEvent.ACTION_MOVE -> {
            // 重新绘制大圆位置
           ...

          }
          MotionEvent.ACTION_UP -> {
                          // 清空bitMap图片
            dragView.upDataBitMap(null, bitMap.width.toFloat())
          }
       }
       true
     }
   }
```

绘制

```
#DragView.kt

fun void onDraw(canvas: Canvas){
 // 绘制小圆
 // 绘制大圆

 // 绘制view中的bitMap
```

```
        bitMap?.let {
            canvas.drawBitmap(it,
                    bigPointF.x - it.width / 2f,
                    bigPointF.y - it.height / 2f, paint)
        }

        // 绘制辅助圆
    }

    var bitMap: Bitmap? = null
    var bitMapWidth = 0f
    fun upDataBitMap(bitMap: Bitmap?, bitMapWidth: Float) {
        this.bitMap = bitMap
        this.bitMapWidth = bitMapWidth
        invalidate()
    }
```

这里的bitMapWidth现在还不用,后面会用到.




## 回弹效果

回弹代码已经写好了,只需要调用一下即可

```
# BlogDragBubbleUtil.kt

MotionEvent.ACTION_UP -> {

  /// 判断大圆是否在辅助圆内
   if (dragView.isContains()) {

    // 回弹效果
    dragView.bigAnimator().run {
     start()

     doOnEnd { // 结束回调
      // 显示View
      view.visibility = View.VISIBLE
      // 删除
      windowManager.removeView(dragView)
      dragView.upDataBitMap(null, bitMap.width.toFloat())
     }
    }
   } else {
    // 爆炸效果

   }

}
```


## 爆炸效果

```
MotionEvent.ACTION_UP -> {

    /// 判断大圆是否在辅助圆内
    if (dragView.isContains()) {
        // 回弹效果
        dragView.bigAnimator().run {
            start()
            doOnEnd { // 结束回调
                // 显示View
                view.visibility = View.VISIBLE
                // 删除
                windowManager.removeView(dragView)
                dragView.upDataBitMap(null, bitMap.width.toFloat())
            }
        }

    } else {
        // 爆炸效果

        // 爆炸之前先清空ViewBitMap
        dragView.upDataBitMap(null, bitMap.width.toFloat())
```

```
            dragView.explodeAnimator.run {
                start() // 开启动画
                doOnEnd {  // 结束动画回调
                    windowManager.removeView(dragView)
                    view.visibility = View.VISIBLE
                }
            }
        }
    }
```


可以看出,基本效果已经完成了,但是还有一点,如果仔细看,

偶尔情况下在大圆回到校园位置的时候,会闪烁一下

解决闪烁问题也很简单,只需要让他在下一帧的时候在进行隐藏或者显示操作即可

那么只需要调用View#postOnAnimation{}方法即可,吧辅助圆去掉,看看现在的效果


此时的效果已经接近完美了~

假如现在换个大一点的控件来看看效果:


可以看出,效果是没问题,但是爆炸范围有一点小,我想控件有多宽,爆炸范围就有多大

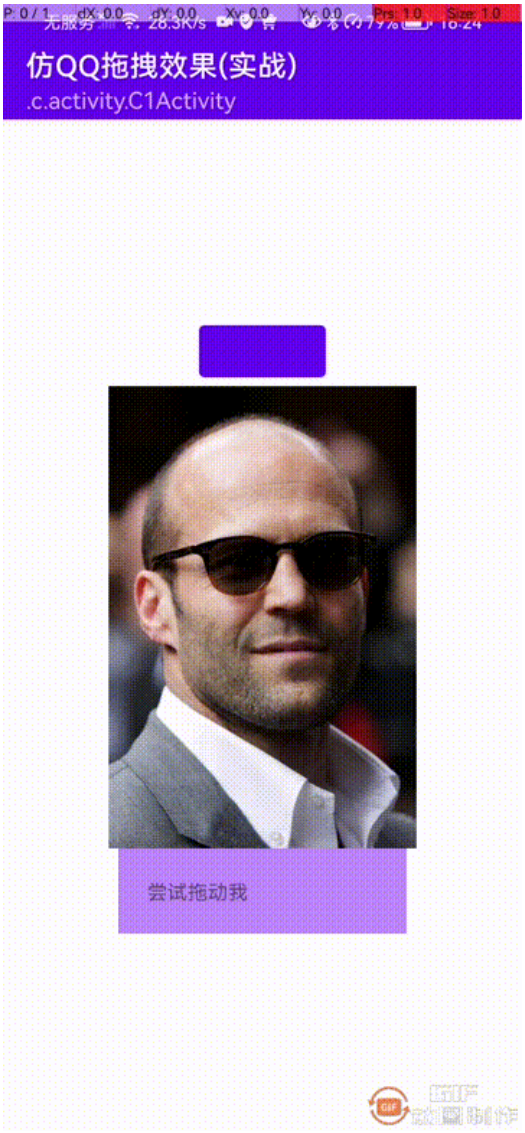
在上面更新View中BitMap图片的时候会传递BitMap的宽度,所以直接设置一下即可

```
private val explodeImages by lazy {
    val list = arrayListOf<Bitmap>()
    val width = bitMapWidth // 设置bitmap 宽度
    list.add(getBitMap(R.mipmap.explode_0, width.toInt()))
     ... 加载20张图片
    list.add(getBitMap(R.mipmap.explode_19, width.toInt()))
    list
}
```

getBitMap是一个加载BitMap的扩展方法

```
fun View.getBitMap(@DrawableRes bitmap: Int = R.mipmap.user, width: Int = 640): Bitmap = let {
    val options = BitmapFactory.Options()
    options.inJustDecodeBounds = true
    BitmapFactory.decodeResource(resources, bitmap)
    options.inJustDecodeBounds = false
    options.inDensity = options.outWidth
```

```
    options.inTargetDensity = width
    BitmapFactory.decodeResource(resources, bitmap, options)
}
```




可以看出,爆照效果会跟随着图片的宽度来变化

但是爆炸的时候会有白底,这完全是因为我不会用ps,真的不会切图... 就这么将就的看吧...

最后在RecyclerView中实战一下!

rv的代码就不看了,太简单了


可以看出,效果还是不对,出错原因子view抢事件没抢过recyclerView,那么只需要在ACTION_DOWN中抢一下事件即可

```
view.setOnTouchListener { v, event ->
       when (event.action) {
  MotionEvent.ACTION_DOWN -> {
    // 和父容器抢焦点
    view.parent.requestDisallowInterceptTouchEvent(true)
  }
         ...
}
```

最终效果达成。

思路参考 ： https://www.jianshu.com/p/9eb9c61e6c8b

完整代码：https://gitee.com/lanyangyangzzz/custom-view-project

推荐阅读：

我的新书，《第一行代码 第3版》已出版！

再看LayoutInflater，这次你可能又会有新的认识

浅谈JCenter即将被停止服务的事件

欢迎关注我的公众号

学习技术或投稿

▼

阅读原文

喜欢此内容的人还喜欢

一种更优雅书写Python代码的方式

Python大数据分析

谷歌大佬开源《Python编程入门》，极致经典，堪称Python入门教程的天花板

程序员森芋


Facebook的新开源项目Velox，有点命运多舛啊。。。

飞总聊IT