自定义View之Path类

> 前言

查看Android总结专题

自定义View总结：

- View基础
- measure方法
- layout方法
- draw方法
- Path类
- Canvas类

1. 简介

- 定义： 路径，即无数个点连接起来的线
- 作用：设置绘制的顺序&区域

> Path 只用于描述顺序&区域，单使用Path无法产生效果

- 应用场景：绘制复杂图形

> Path类封装了由直线和曲线（2,3次贝塞尔曲线）构成的几何路径

2. 基础

2.1 开放路径与闭合路径的区别

闭合路径　　开放路径

3.具体使用

3.1 对象创建

```
// 使用Path首先要new一个Path对象
// Path的起点默认为坐标为(0,0)
Path path = new Path();
// 特别注意：建全局Path对象，在onDraw()按需修改；尽量不要在onDraw()方法里new对象
// 原因：若View频繁刷新，就会频繁创建对象，拖慢刷新速度。
```

3.2 具体方法使用

因为path类的方法都是联合使用，所以下面将一组组方法进行介绍。

第一组 ：设置路径

采用`moveTo()`、`setLastPoint()`、`lineTo()`、`close()`组合

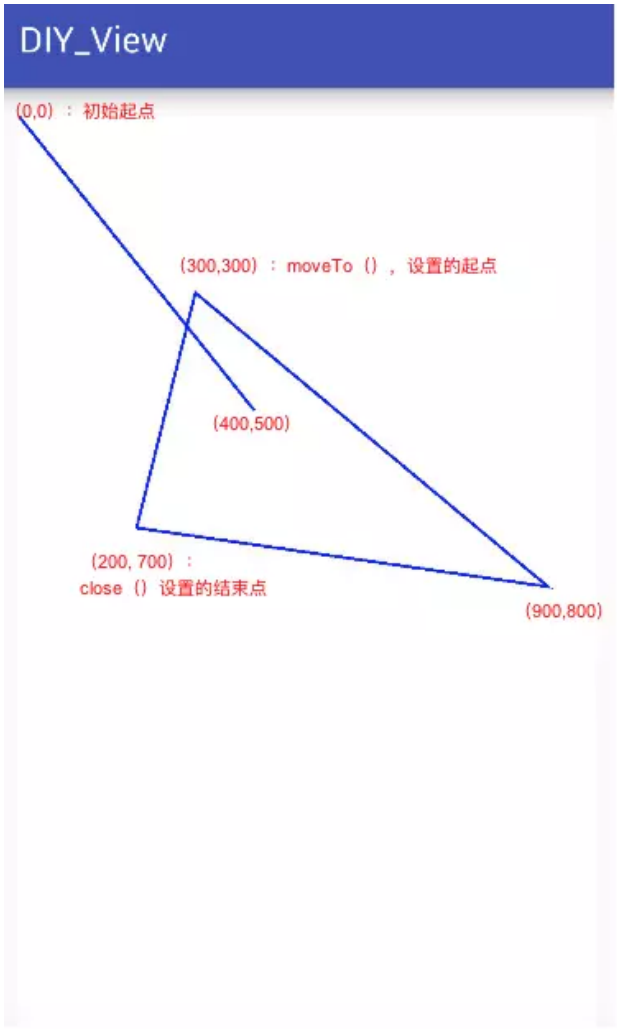
```
// 设置当前点位置
// 后面的路径会从该点开始画
moveTo(float x, float y) ;

// 当前点（上次操作结束的点）会连接该点
// 如果没有进行过操作则默认点为坐标原点。
lineTo(float x, float y)  ;

// 闭合路径，即将当前点和起点连在一起
// 注：如果连接了最后一个点和第一个点仍然无法形成封闭图形，则close什么也不做
close() ;
```

- 可使用`setLastPoint()`设置当前位置(代替`moveTo()`)
- 二者区别：

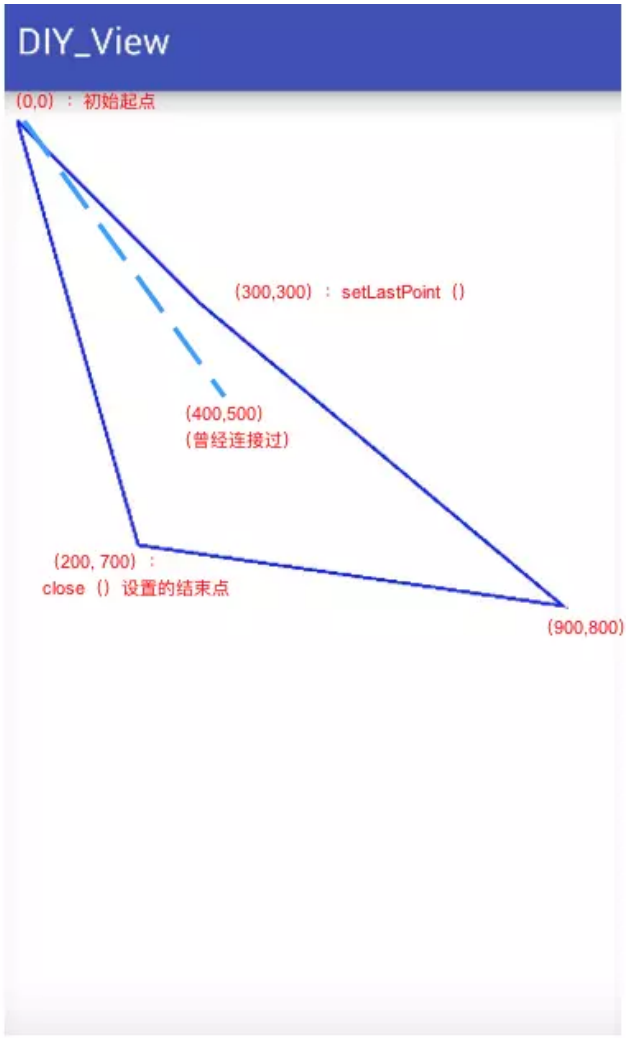
类型	是否影响起点	是否影响之前操作
moveTo()	是	否
setLastPoint()	否	是

实例介绍：

```
        // 使用moveTo ( )
        // 起点默认是(0,0)
        //连接点(400,500)
        path.lineTo(400, 500);

        // 将当前点移动到(300, 300)
        path.moveTo(300, 300) ;

        //连接点(900, 800)
        path.lineTo(900, 800);

        //连接点(200,700)
        path.lineTo(200, 700);

        // 闭合路径，即连接当前点和起点
        // 即连接(200,700)与起点2(300, 300)
        // 注:此时起点已经进行变换
        path.close();

        // 画出路径
        canvas.drawPath(path, mPaint1);

        // 使用setLastPoint ( )
        // 起点默认是(0,0)
        //连接点(400,500)
        path.lineTo(400, 500);

        // 将当前点移动到(300, 300)
        // 会影响之前的操作
        // 但不将此设置为新起点
        path.setLastPoint(300, 300) ;

        //连接点(900,800)
        path.lineTo(900, 800);

        //连接点(200,700)
        path.lineTo(200, 700);

        // 闭合路径，即连接当前点和起点
        // 即连接(200,700)与起点(0，0)
        // 注:起点一直没变化
        path.close();

        // 画出路径
        canvas.drawPath(path, mPaint1);
```



使用moveTo（）



使用setLastPoint（）

关于重置路径

- 重置Path有两个方法：`reset()`和`rewind()`
- 两者区别在于:

类型	是否保留FillType设置	是否保留原有数据结构
Path.reset()	是	否
Path.rewind()	否	是

1. `FillType`影响显示效果；数据结构影响重建速度
2. 一般选择`Path.reset()`

第二组：添加路径

采用`addxxx()`、`arcTo()`组合

- 作用： 在Path路径中添加基本图形
- 具体使用

```
// 添加圆弧
// 方法1
public void addArc (RectF oval, float startAngle, float sweepAngle)

//  startAngle：确定角度的起始位置
//  sweepAngle  ：  确定扫过的角度

    // 方法2
    // 与上面方法唯一不同的是：如果圆弧的起点和上次最后一个坐标点不相同，就连接两个点
    public void arcTo (RectF oval, float startAngle, float sweepAngle)

   // 方法3
   // 参数forceMoveTo：是否将之前路径的结束点设置为圆弧起点
   // true：在新的起点画圆弧，不连接最后一个点与圆弧起点，即与之前路径没有交集（同addArc（））
  // false：在新的起点画圆弧，但会连接之前路径的结束点与圆弧起点，即与之前路径有交集（同arcTo（3参数））
    public void arcTo (RectF oval, float startAngle, float sweepAngle, boolean forceMoveTo)
// 下面会详细说明


  // 加入圆形路径
  // 起点：x轴正方向的0度
  // 其中参数dir：指定绘制时是顺时针还是逆时针:CW为顺时针，  CCW为逆时针
  // 路径起点变为圆在X轴正方向最大的点
  addCircle(float x, float y, float radius, Path.Direction dir)

   // 加入椭圆形路径
  // 其中，参数oval作为椭圆的外切矩形区域
  addOval(RectF oval, Path.Direction dir)

  // 加入矩形路径
  // 路径起点变为矩形的左上角顶点
  addRect(RectF rect, Path.Direction dir)

  //加入圆角矩形路径

  addRoundRect(RectF rect, float rx, float ry, Path.Direction dir)

//  注：添加图形路径后会改变路径的起点
```

第三组：判断路径属性

- 采用`isEmpty()`、`isRect()`、`isConvex()`、`set()`和`offset()`组合
- 具体使用

```
// 判断path中是否包含内容
 public boolean isEmpty ()
// 例子：
```

```
Path path = new Path();
path.isEmpty();  //返回false

 path.lineTo(100,100); // 返回true


// 判断path是否是一个矩形
// 如果是一个矩形的话，会将矩形的信息存放进参数rect中。
public boolean isRect (RectF rect)

// 实例
path.lineTo(0,400);
        path.lineTo(400,400);
        path.lineTo(400,0);
        path.lineTo(0,0);

        RectF rect = new RectF();
        boolean b = path.isRect(rect);  // b返回ture,
        // rect存放矩形参数，具体如下：
        // rect.left = 0
        // rect.top = 0
        // rect.right = 400
        // rect.bottom = 400



// 将新的路径替代现有路径
 public void set (Path src)

        // 实例
        // 设置一矩形路径
        Path path = new Path();
        path.addRect(-200,-200,200,200, Path.Direction.CW);

        // 设置一圆形路径
        Path src = new Path();
        src.addCircle(0,0,100, Path.Direction.CW);

        // 将圆形路径代替矩形路径
        path.set(src);

        // 绘制图形
        canvas.drawPath(path,mPaint);


// 平移路径
// 与Canvas.translate （）平移画布类似


// 方法1
// 参数x,y：平移位置
public void offset (float dx, float dy)

// 方法2
// 参数dst：存储平移后的路径状态，但不影响当前path
// 可通过dst参数绘制存储的路径
```

```
        public void offset (float dx, float dy, Path dst)



 // 为了方便观察,平移坐标系
        canvas.translate(350, 500);

        // path中添加一个圆形(圆心在坐标原点)
        path = new Path();
        path.addCircle(0, 0, 100, Path.Direction.CW);

        // 平移路径并存储平移后的状态
        Path dst = new Path();
        path.offset(400, 0, dst);                    // 平移

        canvas.drawPath(path, mPaint1);              // 绘制path


        // 通过dst绘制平移后的图形(红色)
        mPaint1.setColor(Color.RED);
        canvas.drawPath(dst,mPaint1);

```

第四组：设置路径填充颜色

- 在Android中，有四种填充模式，具体如下（均封装在Path类中）

填充模式	介绍
EVEN_ODD	奇偶规则
INVERSE_EVEN_ODD	反奇偶规则
WINDING	非零环绕数规则
INVERSE_WINDING	反非零环绕数规则

图形是存在方向的（画图=连接点的线=有连接顺序）

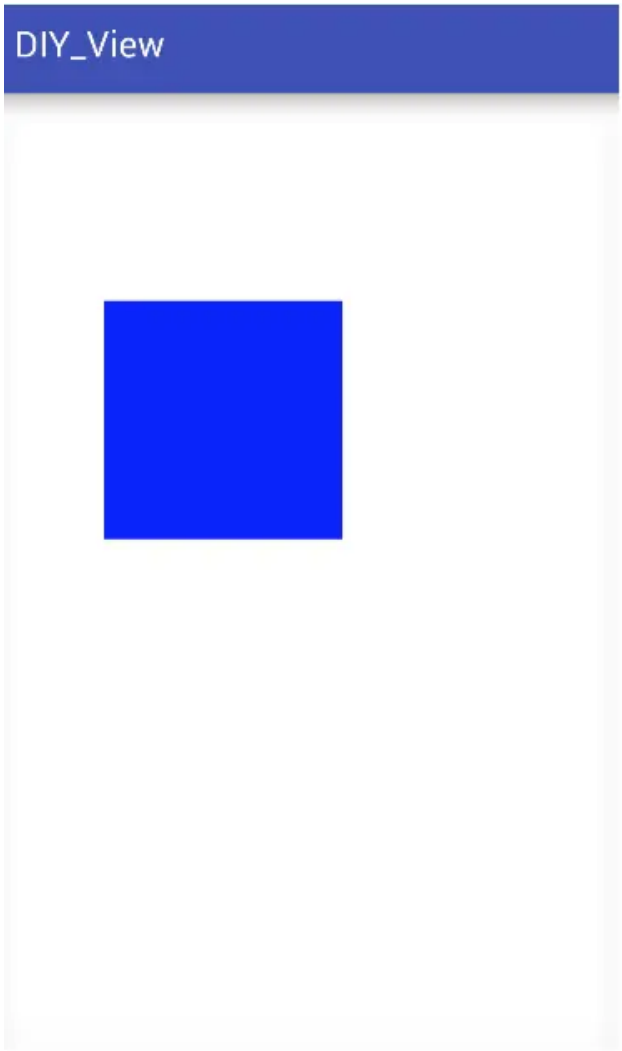
- 具体使用

```
// 设置填充规则
path.setFillType()
// 可填规则
// 1. EVEN_ODD：奇偶规则
// 2. INVERSE_EVEN_ODD：反奇偶规则
// 3. WINDING ：非零环绕数规则
// 4. INVERSE_WINDING：反非零环绕数规则

// 理解奇偶规则和反奇偶规则：填充效果相反
// 举例：对于一个矩形而言，使用奇偶规则会填充矩形内部，而使用反奇偶规则会填充矩形外部（下面会举例说明）

// 获取当前填充规则

```

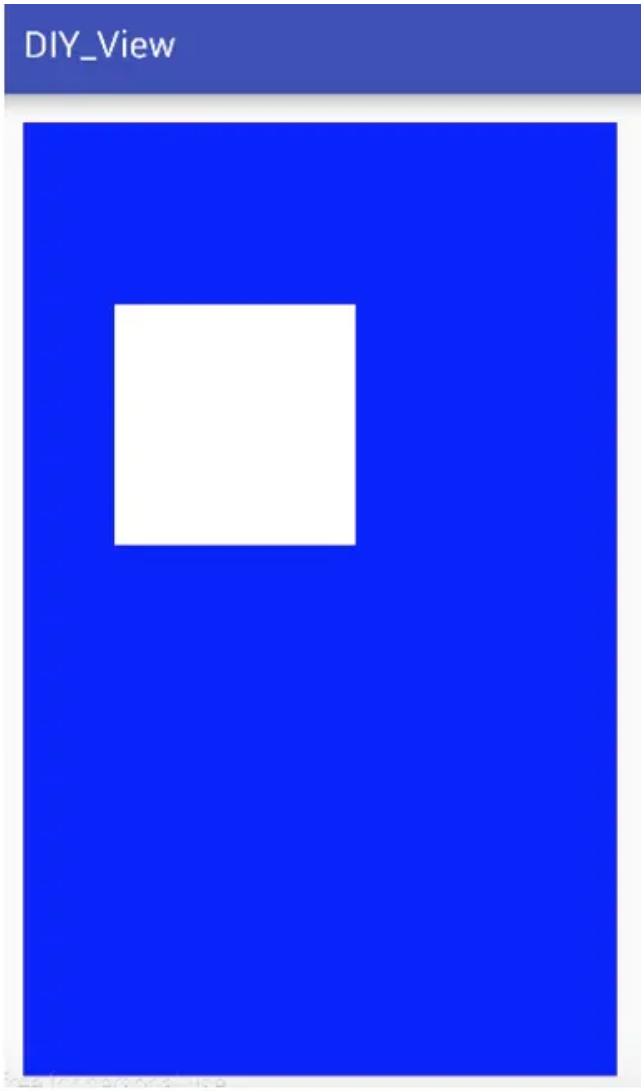
```
path.getFillType()

// 判断是否是反向(INVERSE)规则
path.isInverseFillType()

// 切换填充规则(即原有规则与反向规则之间相互切换)
path.toggleInverseFillType()
```

实例1：（奇偶规则）

```

 // 为了方便观察,平移坐标系
        canvas.translate(350, 500);

        // 在Path中添加一个矩形
        path.addRect(-200, -200, 200, 200, Path.Direction.CW);

        // 设置Path填充模式为 奇偶规则
        path.setFillType(Path.FillType.EVEN_ODD);

        // 反奇偶规则
        // path.setFillType(Path.FillType.INVERSE_EVEN_ODD);

        // 画出路径
        canvas.drawPath(path, mPaint1);
```



奇偶规则



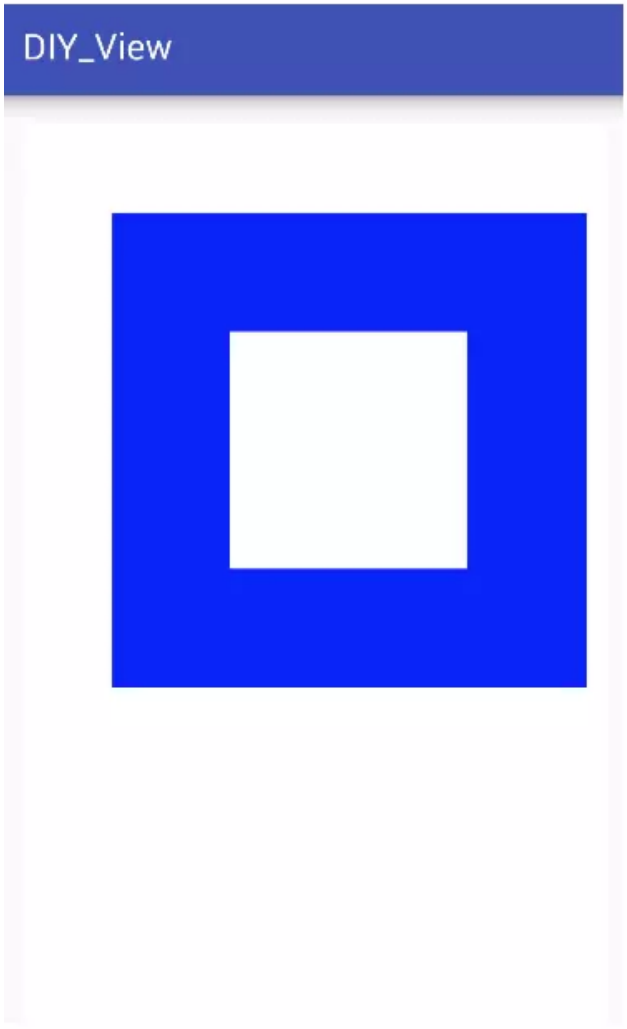
反奇偶规则

实例2：(非零环绕规则)

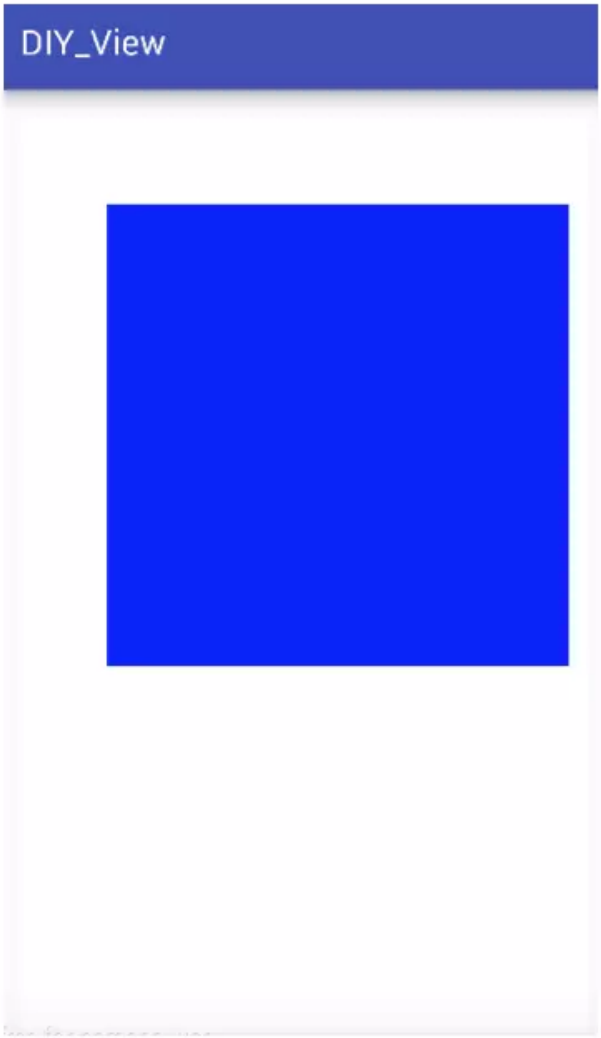
```
    // 为了方便观察,平移坐标系
        canvas.translate(550, 550);
        // 在路径中添加大正方形
        // 逆时针
        path.addRect(-400, -400, 400, 400, Path.Direction.CCW);

        // 在路径中添加小正方形
        // 顺时针
//        path.addRect(-200, -200, 200, 200, Path.Direction.CW);
//          设置为逆时针
          path.addRect(-200, -200, 200, 200, Path.Direction.CCW);


        // 设置Path填充模式为非零环绕规则
        path.setFillType(Path.FillType.WINDING);
        // 设置反非零环绕数规则
        // path.setFillType(Path.FillType.INVERSE_WINDING);

```

```
20          // 绘制Path
            canvas.drawPath(path, mPaint1);
```

非零环绕规则

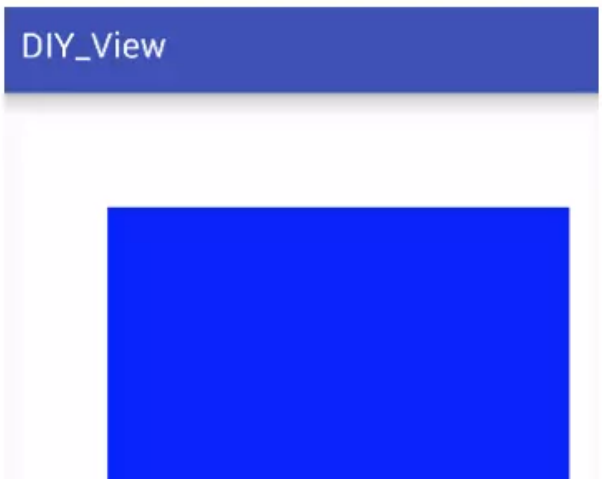


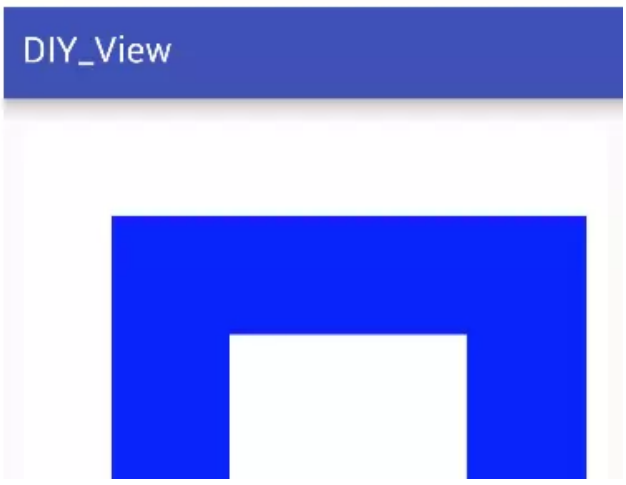
内外图形方向相反



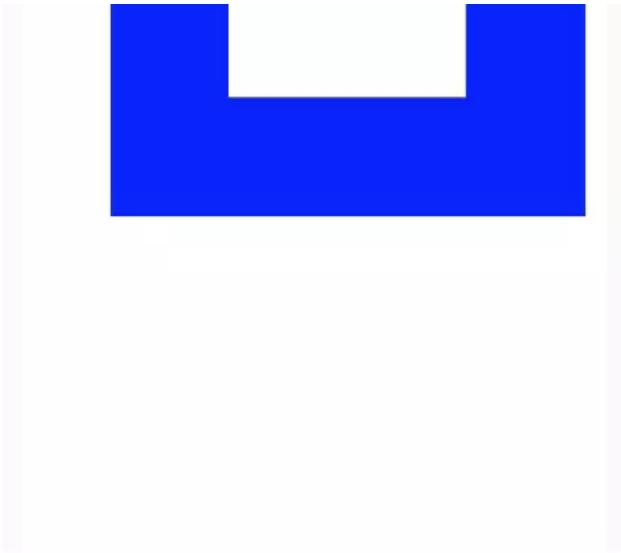
内外图形方向相同

相反零环绕规则





内外图形方向相反



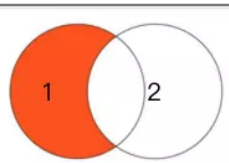
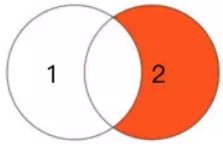
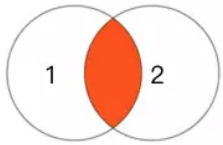
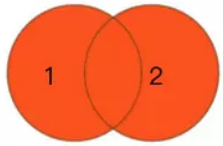
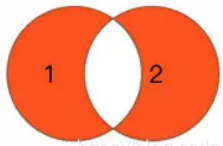
内外图形方向相同

第五组：布尔操作

- 作用：两个路径Path之间的运算
- 应用场景：用简单的图形通过特定规则合成相对复杂的图形
- 具体使用：

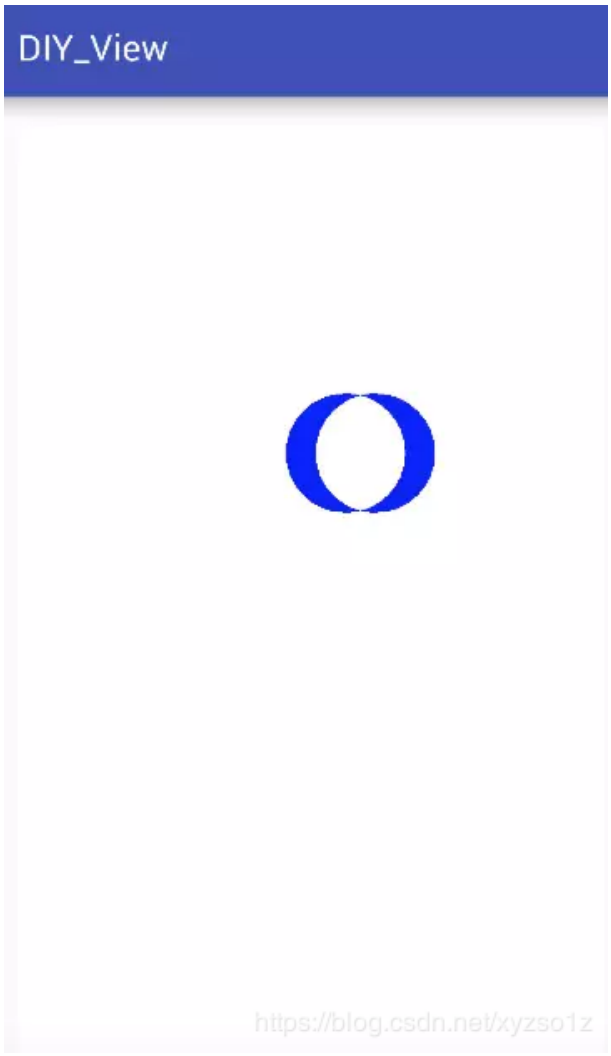
```
// 方法1
    boolean op (Path path, Path.Op op)
// 举例
// 对 path1 和 path2 执行布尔运算，运算方式由第二个参数指定
// 运算结果存入到path1中。
    path1.op(path2, Path.Op.DIFFERENCE);


// 方法2
    boolean op (Path path1, Path path2, Path.Op op)
  // 举例
    // 对 path1 和 path2 执行布尔运算，运算方式由第三个参数指定
    // 运算结果存入到path3中。
    path3.op(path1, path2, Path.Op.DIFFERENCE)
```

之间的运算方式（即Path.op参数）如下：

参数	说明	示意图
DIFFERENCE	path1不同于path2的区域	
REVERSEDIFFERENCE	path2不同于path1的区域	
INTERSECT	（交集）path1与path2相交区域	
UNION	（并集）path1与path2的和	
XOR	（异或）path1与path2和并减去重叠的部分	

举例：

```
// 为了方便观察,平移坐标系
      canvas.translate(550, 550);

      // 画两个圆
      // 圆1:圆心 = (0,0),半径 = 100
      // 圆2:圆心 = (50,0),半径 = 100
      path1.addCircle(0, 0, 100, Path.Direction.CW);
      path2.addCircle(50, 0,100, Path.Direction.CW);

      // 取两个路径的异或集
      path1.op(path2, Path.Op.XOR);
      // 画出路径
      canvas.drawPath(path1, mPaint1);
```



4.贝塞尔曲线

- 定义：计算曲线的数学公式
- 作用：计算并表示曲线

> 任何一条曲线都可以用贝赛尔曲线表示

- 具体使用：贝塞尔曲线可**通过1数据点和若干个控制点描述**

> 1. 数据点： 指路径的起始点和终止点;
> 2. 控制点：决定路径的弯曲轨迹;
> 3. n+1阶贝塞尔=有n个控制点;
> 4. 1阶=一条直线，高阶可以拆解为多条低阶曲线;

Canvas提供了画二阶 & 三阶贝塞尔曲线的方法，下面是具体方法：

```
1
2  // 绘制二阶贝塞尔曲线
3  //  (x1,y1)为控制点，(x2,y2)为终点
4  quadTo(float x1, float y1, float x2, float y2)
5  //  (x1,y1)为控制点距离起点的偏移量，(x2,y2)为终点距离起点的偏移量
6  rQuadTo(float x1, float y1, float x2, float y2)
7
8  // 绘制三阶贝塞尔曲线
9  // (x1,y1),(x2,y2)为控制点，(x3,y3)为终点
10 cubicTo(float x1, float y1, float x2, float y2, float x3, float y3)
11 // (x1,y1)，(x2,y2)为控制点距离起点的偏移量，(x3,y3)为终点距离起点的偏移量
12 rCubicTo(float x1, float y1, float x2, float y2, float x3, float y3)
13
```

xyzso1z
原创文章 80　获赞 40　访问量 2万+