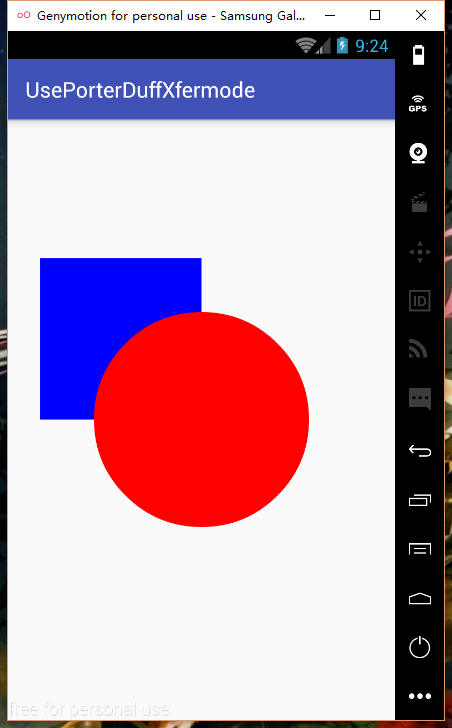
楔子

我们在自定义的过程，经常会遇到多个图形相交的问题（如下图），那么系统是如何处理图片相交部分的绘制的呢？



View的基本框架(之后的代码都是基于该View)：

public class PorterDuffXfermodeView extends View {

private final Paint mPaint = new Paint();

private Bitmap mBitmap;

private Bitmap mOut;

private int mViewWidth;

private int mViewHeight;

public PorterDuffXfermodeView(Context context) {

this(context,null);

}

public PorterDuffXfermodeView(Context context, AttributeSet attrs) {

this(context, attrs,0);

}

public PorterDuffXfermodeView(Context context, AttributeSet attrs, int defStyleAttr) {

super(context, attrs, defStyleAttr);

initWidget();

}

private void initWidget(){

//初始化画笔

mPaint.setStyle(Paint.Style.FILL);

mPaint.setAntiAlias(true);

mPaint.setDither(true);

}

@Override

protected void onSizeChanged(int w, int h, int oldw, int oldh) {

super.onSizeChanged(w, h, oldw, oldh);

mViewWidth = w;

mViewHeight = h;

}

@Override

protected void onDraw(Canvas canvas) {

super.onDraw(canvas);

}

}

实现代码：

//自定义View

@Override

protected void onDraw(Canvas canvas) {

super.onDraw(canvas);

canvas.translate(mViewWidth/2,mViewHeight/2);

firstExample(canvas);

}

/\*\*

\* 制作具有相交部分的圆和正方形

\*/

private void firstExample(Canvas canvas){

//绘制一个正方型

mPaint.setColor(Color.BLUE);

canvas.drawRect(-300,-300,0,0,mPaint);

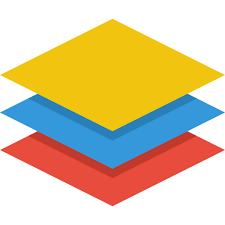
//绘制一个圆

mPaint.setColor(Color.RED);

canvas.drawCircle(0,0,200,mPaint);

}

首先我们要知道系统是如何绘制图形的，系统每绘制一个图形（如：canvas.drawXxx()）就代表创建了一个图层，那么什么叫做图层?

我们看一张图就明白了   
   
1、图片中的三个颜色分别代表一个图层，每个图层的内容就是绘制的图形。   
2、系统默认是图层向上累加绘制，也就是红色是最先绘制的，蓝色是之后绘制的，黄色是最后绘制的。所以图形相交的部分，就被上层的图层的内容给掩盖了。

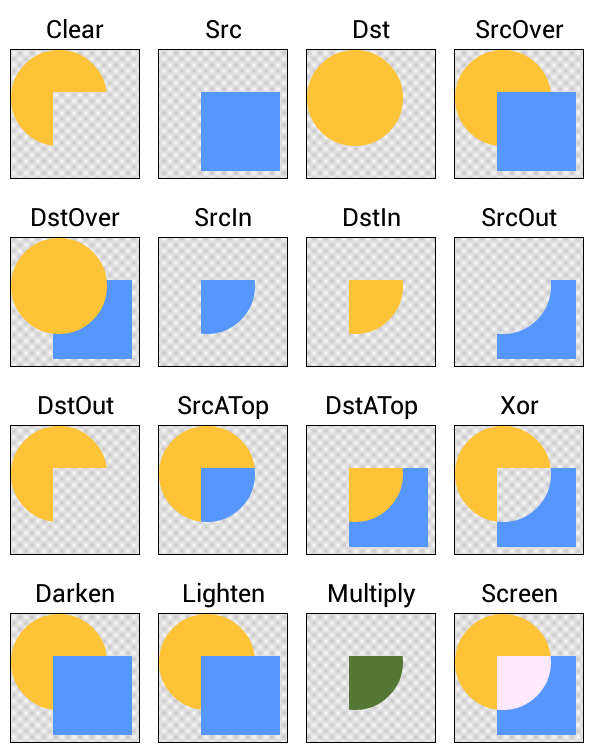
既然明白了图层的概念，我们就可以回到正题，如何处理图层的相交部分。[**Android**](http://lib.csdn.net/base/android)为我们提供了**PorterDuffXfermode**这个类，来处理关于图层相交的问题。

# PorterDuffXfermode的使用

## PorterDuffXfermode能够实现的功能

首先我们来看一下PorterDuffXfermode有哪些功能

**注：(dst表示先绘制的图，src表示后绘制的图)**



稍微简单的解释一下常用的功能：   
凡是带有IN的表示：取两个图层的相交部分，关于相交部分显示什么内容有DST和SRC决定。   
凡是带有OUT的表示：取两个图层中另一方不相交的部分。   
凡是带有OVER的表示：当俩个图层存在相交部分时，显示哪个图层的内容

## PorterDuffXfermode的简单使用

作用一：图层的交换   
任务：将第一幅图的正方形显示在顶部，圆形显示在底部。

首先如何创建PorterDuffXfermode。

//构造方法

/\*\*

\* PorterDuff.Mode：这是一个枚举类，枚举类的参数为上面的示意图

\*/

PorterDuffXfermode(PorterDuff.Mode mode)

然后，将创建好的PorterDuffXfermode放入Paint中

mPaint.setXfermode(new PorterDuffXfermode(mode))

为什么是将模式放入到画笔中0 0，这是表示画笔当遇到图形相交的问题的时候，按照这种方式来解决。

代码的实现：

private void secondExample(Canvas canvas){

//绘制一个正方型

mPaint.setColor(Color.BLUE);

canvas.drawRect(-300,-300,0,0,mPaint);

//设置相交时候，图层显示的模式(表示相交部分显示前一个图层的内容)

mPaint.setXfermode(new PorterDuffXfermode(PorterDuff.Mode.DST\_OVER));

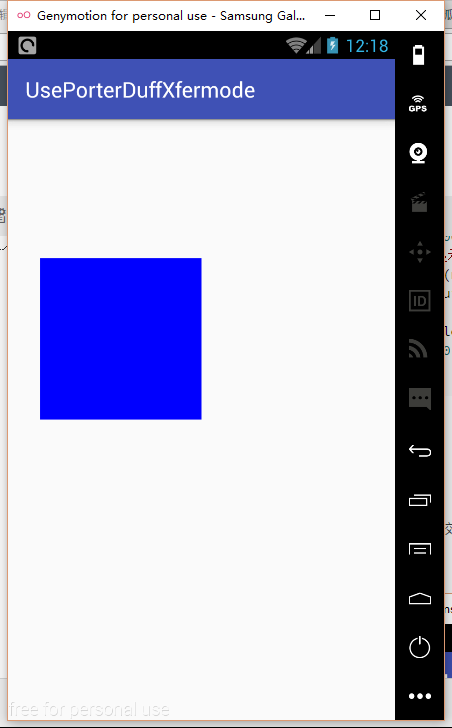
//绘制一个圆

mPaint.setColor(Color.RED);

canvas.drawCircle(0,0,200,mPaint);

}

效果图 ：



我们发现，竟然红色的圆消失了 - -，什么鬼跟说好的不一样呀！！。   
原因是这种直接使用PorterDuffXfermode的用法是错误的- -，真是糟糕的体验，哪里错了，原理上完全没问题好吧。

（**警告：PorterDuffXfermode的第一道坑**）   
**正确的使用方法**：首先需要创造一个Bitmap，然后首先在这个Bitmap上绘制完成之后，再将这个Bitmap通过canvas显示在View上。

**正确代码**：

private void thirdExample(Canvas canvas){

//创建一个Bitmap

Bitmap out = Bitmap.createBitmap(600,600, Bitmap.Config.ARGB\_8888);

//创建该Bitmap的画布

Canvas bitmapCanvas = new Canvas(out);

//绘制一个正方型

mPaint.setColor(Color.BLUE);

bitmapCanvas.drawRect(0,0,300,300,mPaint);

//设置相交时候，图层显示的模式(表示当相交的时候，圆形为先绘制的图形)

mPaint.setXfermode(new PorterDuffXfermode(PorterDuff.Mode.DST\_OVER));

//绘制一个圆

mPaint.setColor(Color.RED);

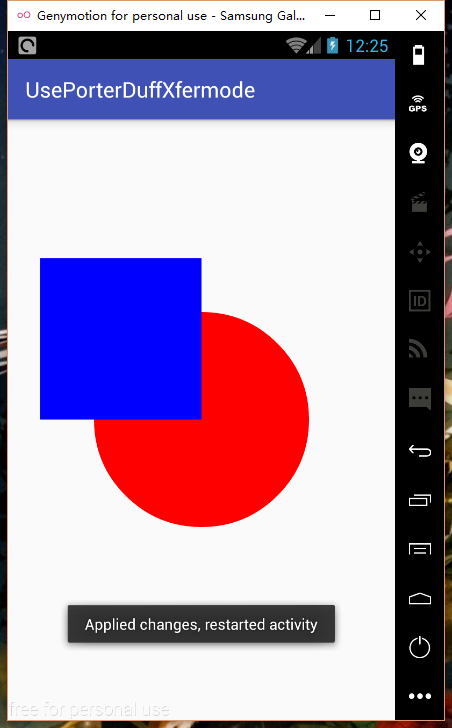
bitmapCanvas.drawCircle(300,300,200,mPaint);

//最后，将完成的图片绘制在View上

canvas.drawBitmap(out,-300,-300,null);

}

效果图：

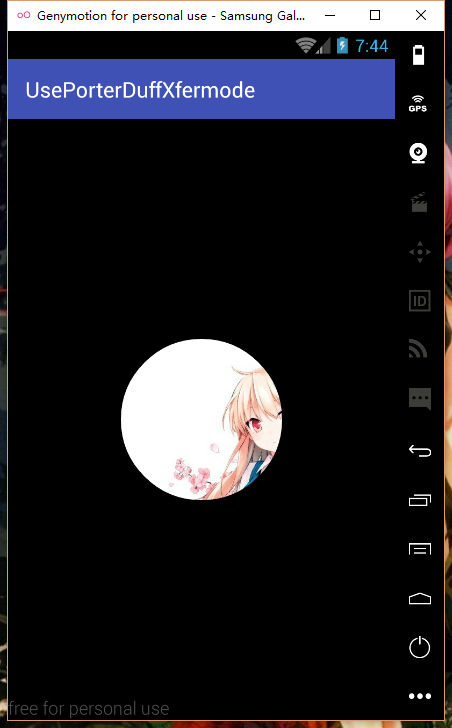


天哪，真是太麻烦了- - ，直接让圆形和方形的绘制顺序调换一下，才是最快最省时间的方法 ~~。当然对与这个例子通过调换执行顺序是最快的，但是毕竟我们是来学习使用技巧的。

## 实现圆框图片

作用二：实现两张图相交部分显示的效果

**效果图**：



原理说明：首先创建一个圆形，这个圆形是用来设置图片显示的区域。之后获取图片资源（图片的大小最好比设定的圆形大）。然后，设置Xfermode为SRC\_IN就表示，当圆形和图片相交的时候，截取相交部分（也就是截取圆形），相交部分显示后一个图层的内容（也就是图片）

**代码展示**：

private void forthExample(Canvas canvas){

//创建自定义的Bitmap

Bitmap out = Bitmap.createBitmap(300,300, Bitmap.Config.ARGB\_8888);

//获取图片

Bitmap bitmap = BitmapFactory.decodeResource(getResources(), R.mipmap.picture);

//获取Bitmap的画笔

Canvas bitmapCanvas = new Canvas(out);

//在Bitmap上绘制一个圆形

bitmapCanvas.drawCircle(150,150,150,mPaint);

//设置显示后画图形的交集

mPaint.setXfermode(new PorterDuffXfermode(PorterDuff.Mode.SRC\_IN));

//绘制需要显示的图片

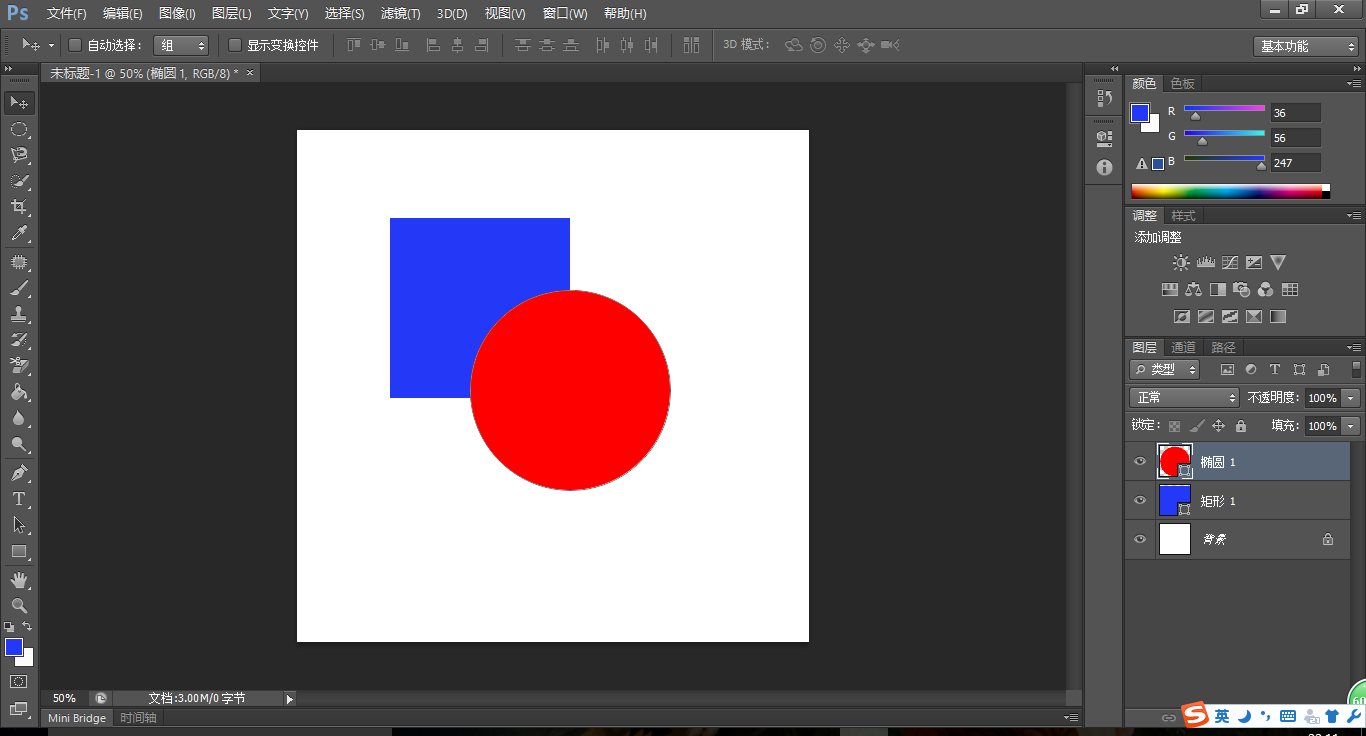
bitmapCanvas.drawBitmap(bitmap,0,0,mPaint);

canvas.drawBitmap(out,-150,-150,null);

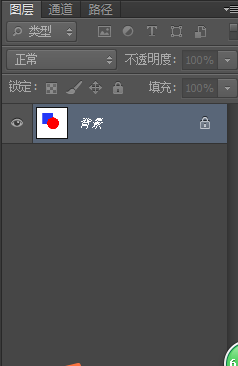
}

## 补充：图层的合并

在制作刮刮卡我们还需要补充一个知识，在开始的位置我曾讲到绘制一次图形就相当于创建一个图层。



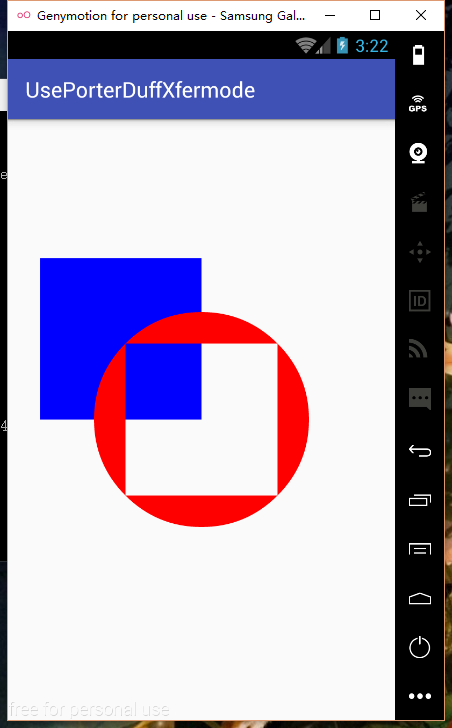
其实实际上还有一个步骤，当使用canvas完成一次绘制过程后，就会将当前图层，和上一个图层合并为一个图层。也就是说，我们首先创建了一个图层，并在其上画了个正方形，然后我们又创建了一个图层，并在其上绘制了一个圆形。当圆形绘制完成的时候，两个图层合并成为了一个图层。



那么这有什么后果呢？

合并之后就表示，相交部分不再是红色覆盖在蓝色上的上下层关系。而是相交部分是红色代替了。

如果不合并图层的话，那么我创建了第三个图层，是圆的内接矩形，并使用DST\_OUT（表示挖出红色圆的内接矩形）那么之后，倒影出得应该是这样的图形。



代码：

private void fifthExample(Canvas canvas){

//首先在View上绘制正方形

mPaint.setColor(Color.BLUE);

canvas.drawRect(-300,-300,0,0,mPaint);

//创建Bitmap

Bitmap out = Bitmap.createBitmap(400,400, Bitmap.Config.ARGB\_8888);

Canvas bitmapCanvas = new Canvas(out);

//在Bitmap上绘制圆

mPaint.setColor(Color.RED);

bitmapCanvas.drawCircle(200,200,200,mPaint);

//设置模式为DST\_OUT

mPaint.setXfermode(new PorterDuffXfermode(PorterDuff.Mode.DST\_OUT));

//圆的内接正方形的参数

float squareLeft = 200 - 100 \* (float)Math.sqrt(2);

float squareTop = squareLeft;

float squareRight = squareLeft + 200 \* (float)Math.sqrt(2);

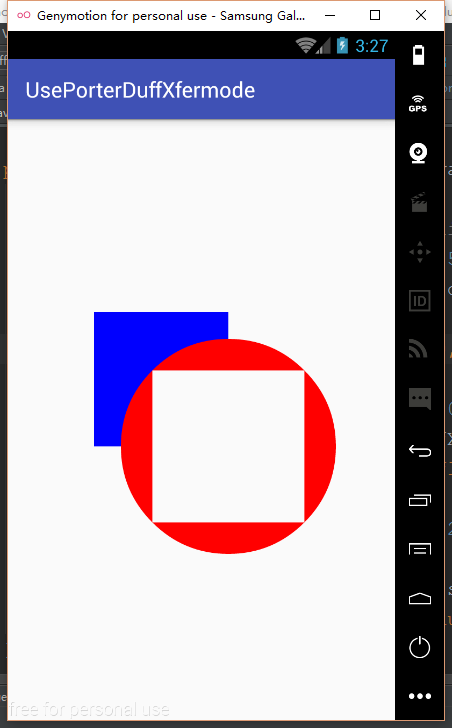
float squareBottom = squareRight;

//取圆与内接正方形不相交的部分 bitmapCanvas.drawRect(squareLeft,squareTop,squareRight,squareBottom,mPaint);

//绘制在View上

canvas.drawBitmap(out,-200,-200,null);

}



private void sixthExample(Canvas canvas){

Bitmap out = Bitmap.createBitmap(500,500, Bitmap.Config.ARGB\_8888);

Canvas bitmapCanvas = new Canvas(out);

//在Bitmap上绘制正方形（这是与上面代码区别的部分）

mPaint.setColor(Color.BLUE);

bitmapCanvas.drawRect(0,0,250,250,mPaint);

mPaint.setColor(Color.RED);

bitmapCanvas.drawCircle(250,250,200,mPaint);

mPaint.setXfermode(new PorterDuffXfermode(PorterDuff.Mode.DST\_OUT));

float squareLeft = 250 - 100 \* (float)Math.sqrt(2);

float squareTop = squareLeft;

float squareRight = squareLeft + 200 \* (float)Math.sqrt(2);

float squareBottom = squareRight;

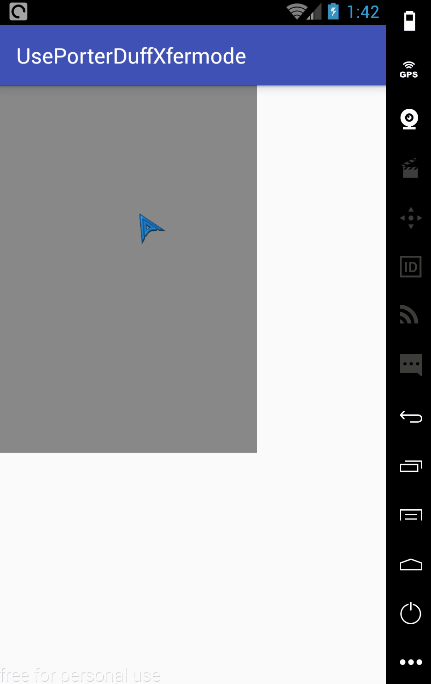
bitmapCanvas.drawRect(squareLeft,squareTop,squareRight,squareBottom,mPaint);

canvas.drawBitmap(out,-200,-200,null);

}

## 大招：实现刮刮卡的效果

效果图：



原理：首先获取遮盖的图片，之后再创建遮盖在图片上的蒙版。最后通过点击事件，设置擦除的路径，制作出路径和蒙版合并的遮罩层。

**代码**：

public class ScratchView extends View {

private final Paint mPaint = new Paint();

private Bitmap mContentBitmap;

private Bitmap mMaskBitmap;

private Canvas mBitmapCanvas;

private final Path mPath = new Path();

private int mViewWidth;

private int mViewHeight;

private int mBitmapWidth;

private int mBitmapHeight;

public ScratchView(Context context) {

this(context,null);

}

public ScratchView(Context context, AttributeSet attrs) {

this(context, attrs,0);

}

public ScratchView(Context context, AttributeSet attrs, int defStyleAttr) {

super(context, attrs, defStyleAttr);

initWidget();

}

private void initWidget(){

//初始化画笔

mPaint.setStyle(Paint.Style.STROKE);

mPaint.setStrokeWidth(20);

mPaint.setAntiAlias(true);

mPaint.setDither(true);

mPaint.setStrokeCap(Paint.Cap.ROUND);

mPaint.setColor(Color.TRANSPARENT);

mPaint.setXfermode(new PorterDuffXfermode(PorterDuff.Mode.SRC\_IN));

//初始化，刮刮卡的内容

mContentBitmap = BitmapFactory.

decodeResource(getResources(), R.mipmap.picture);

mBitmapWidth = mContentBitmap.getWidth();

mBitmapHeight = mContentBitmap.getHeight();

//初始化刮刮卡的遮盖效果

mMaskBitmap = Bitmap.createBitmap(mBitmapWidth,mBitmapHeight, Bitmap.Config.ARGB\_8888);

mBitmapCanvas = new Canvas(mMaskBitmap);

//设置灰色的遮盖层

mBitmapCanvas.drawColor(Color.GRAY);

}

@Override

protected void onSizeChanged(int w, int h, int oldw, int oldh) {

super.onSizeChanged(w, h, oldw, oldh);

mViewWidth = w;

mViewHeight = h;

}

@Override

protected void onDraw(Canvas canvas) {

super.onDraw(canvas);

//绘制图片

canvas.drawBitmap(mContentBitmap,0,0, null);

//绘制蒙版

canvas.drawBitmap(mMaskBitmap,0,0,null);

}

@Override

public boolean onTouchEvent(MotionEvent event) {

int x = (int) event.getX();

int y = (int) event.getY();

switch (event.getAction()){

case MotionEvent.ACTION\_DOWN:

mPath.moveTo(x,y);

break;

case MotionEvent.ACTION\_MOVE:

mPath.lineTo(x,y);

break;

case MotionEvent.ACTION\_UP:

mPath.lineTo(x,y);

break;

}

//设置擦除的路径

mBitmapCanvas.drawPath(mPath,mPaint);

//重绘

invalidate();

return true;

}

}

**但是我们知道在自定义View中不光有canvas，还有path，如何在Path类上，使用类似集合这样的功能呢？**

# Path使用交、并、部的效果

[Path遇到相交情况的解决办法](https://github.com/GcsSloop/AndroidNote/blob/master/CustomView/Advance/%5B07%5DPath_Over.md)