HenCoder Android 开发进阶 自定义 View 1-4 Canvas 对绘制的辅助 clipXXX() 和 Matrix

这期是 HenCoder 自定义绘制的第 1-4 期：Canvas 对绘制的辅助——范围裁切和几何变换。

简介

1 范围裁切

范围裁切有两个方法： `clipRect()` 和 `clipPath()`。裁切方法之后的绘制代码，都会被限制在裁切范围内。

1.1 clipRect()

使用很简单，直接应用：

```
canvas.clipRect(left, top, right, bottom);
canvas.drawBitmap(bitmap, x, y, paint);
```

原内容

裁切后

记得要加上 `Canvas.save()` 和 `Canvas.restore()` 来及时恢复绘制范围，所以完整代码是这样的：

```
canvas.save();
canvas.clipRect(left, top, right, bottom);
canvas.drawBitmap(bitmap, x, y, paint);
canvas.restore();
```

1.2 clipPath()

其实和 clipRect() 用法完全一样，只是把参数换成了 `Path` ，所以能裁切的形状更多一些：

```
canvas.save();
canvas.clipPath(path1);
canvas.drawBitmap(bitmap, point1.x, point1.y, paint);
canvas.restore();

canvas.save();
canvas.clipPath(path2);
canvas.drawBitmap(bitmap, point2.x, point2.y, paint);
canvas.restore();
```

2 几何变换

几何变换的使用大概分为三类：

1. 使用 `Canvas` 来做常见的二维变换；
2. 使用 `Matrix` 来做常见和不常见的二维变换；
3. 使用 `Camera` 来做三维变换。

2.1 使用 Canvas 来做常见的二维变换：

2.1.1 Canvas.translate(float dx, float dy) 平移

参数里的 `dx` 和 `dy` 表示横向和纵向的位移。

```
canvas.save();
canvas.translate(200, 0);
canvas.drawBitmap(bitmap, x, y, paint);
canvas.restore();
```

原内容

平移后

> 好吧这个从截图并不能看出什么，那你就用心去感受吧

2.1.2 Canvas.rotate(float degrees, float px, float py) 旋转

参数里的 degrees 是旋转角度，单位是度（也就是一周有 360° 的那个单位），方向是顺时针为正向； px 和 py 是轴心的位置。

```
canvas.save();
canvas.rotate(45, centerX, centerY);
canvas.drawBitmap(bitmap, x, y, paint);
canvas.restore();
```

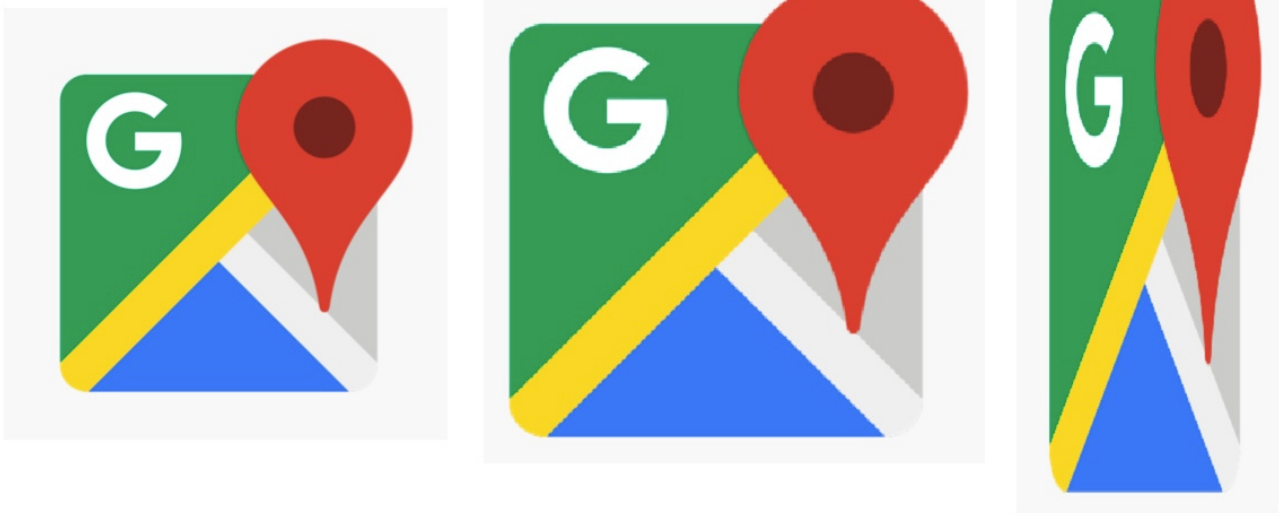
原内容

旋转后

2.1.3 Canvas.scale(float sx, float sy, float px, float py) 放缩

参数里的 sx sy 是横向和纵向的放缩倍数； px py 是放缩的轴心。

```
canvas.save();
canvas.scale(1.3f, 1.3f, x + bitmapWidth / 2, y + bitmapHeight / 2)
canvas.drawBitmap(bitmap, x, y, paint);
canvas.restore();
```

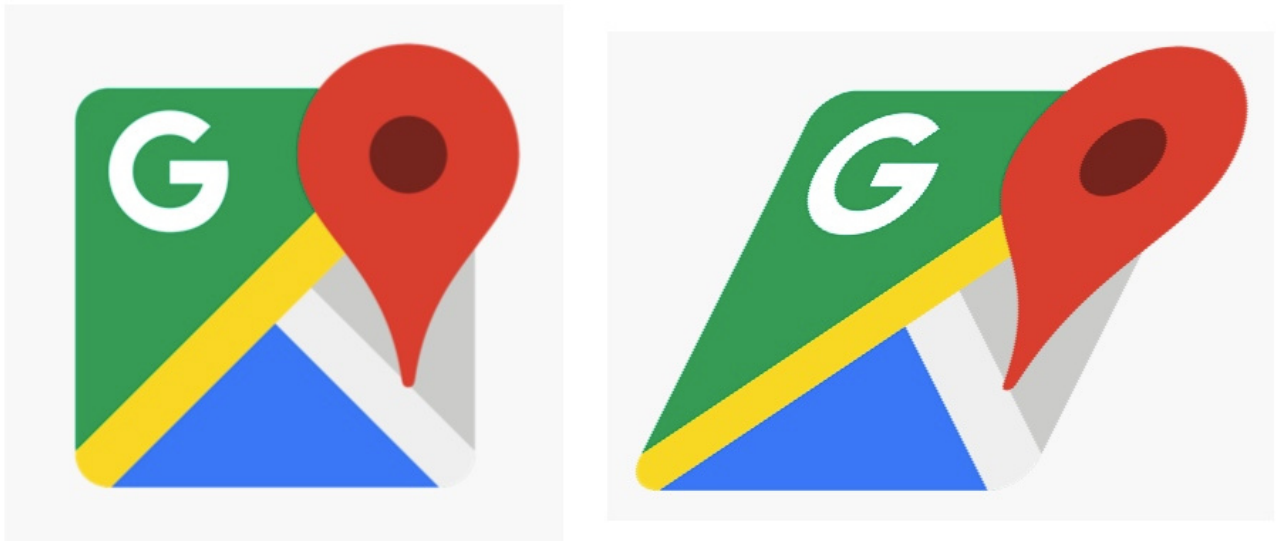


原内容　　等比例放大　　不等比例放缩

2.1.4 skew(float sx, float sy) 错切

参数里的 `sx` 和 `sy` 是 x 方向和 y 方向的错切系数。

```
canvas.save();
canvas.skew(0, 0.5f);
canvas.drawBitmap(bitmap, x, y, paint);
canvas.restore();
```



原内容　　错切后

2.2 使用 Matrix 来做变换

2.2.1 使用 Matrix 来做常见变换

Matrix 做常见变换的方式：

1. 创建 Matrix 对象；
2. 调用 Matrix 的 pre/postTranslate/Rotate/Scale/Skew() 方法来设置几何变换；
3. 使用 Canvas.setMatrix(matrix) 或 Canvas.concat(matrix) 来把几何变换应用到 Canvas 。

```
Matrix matrix = new Matrix();

...

matrix.reset();
matrix.postTranslate();
matrix.postRotate();

canvas.save();
canvas.concat(matrix);
canvas.drawBitmap(bitmap, x, y, paint);
canvas.restore();
```

效果就不放图了，和 Canvas 是一样的。

把 Matrix 应用到 Canvas 有两个方法： Canvas.setMatrix(matrix) 和 Canvas.concat(matrix) 。

1. Canvas.setMatrix(matrix)：用 Matrix 直接替换 Canvas 当前的变换矩阵，即抛弃 Canvas 当前的变换，改用 Matrix 的变换（注：根据下面评论里以及我在微信公众号中收到的反馈，不同的系统中 setMatrix(matrix) 的行为可能不一致，所以还是尽量用 concat(matrix) 吧）；

2. `Canvas.concat(matrix)`：用 `Canvas` 当前的变换矩阵和 `Matrix` 相乘，即基于 `Canvas` 当前的变换，叠加上 `Matrix` 中的变换。

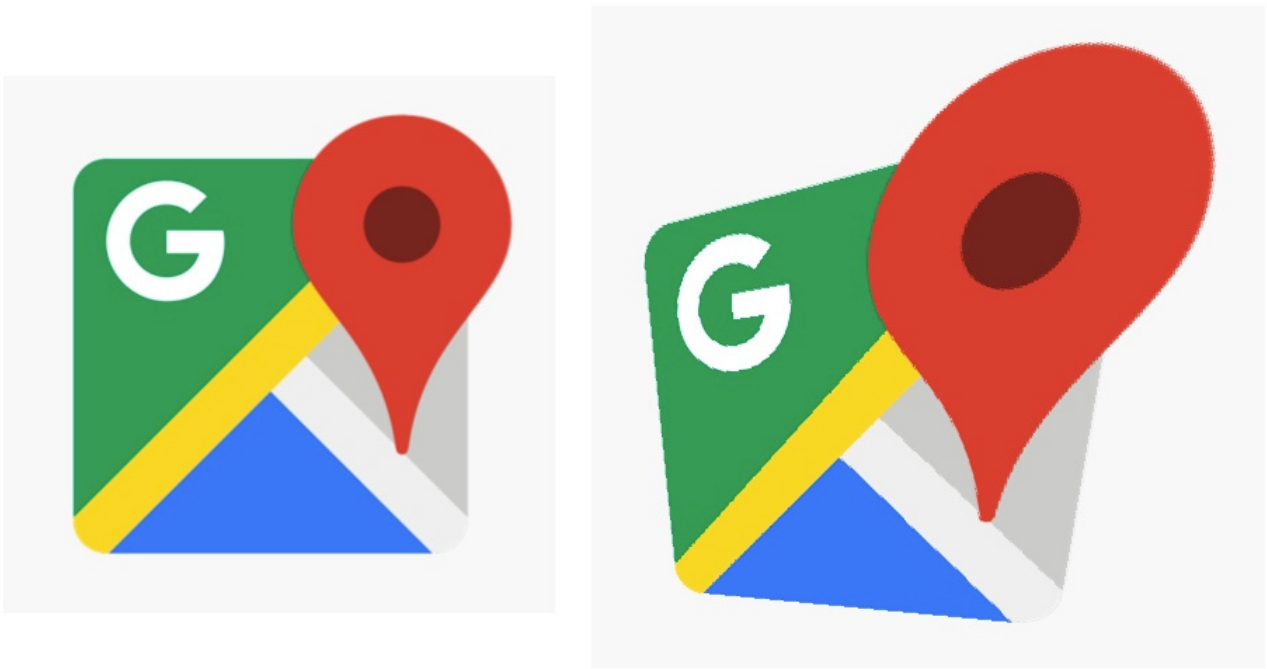
2.2.2 使用 Matrix 来做自定义变换

`Matrix` 的自定义变换使用的是 `setPolyToPoly()` 方法。

2.2.2.1 Matrix.setPolyToPoly(float[] src, int srcIndex, float[] dst, int dstIndex, int pointCount) 用点对点映射的方式设置变换

`poly` 就是「多」的意思。`setPolyToPoly()` 的作用是通过多点的映射的方式来直接设置变换。「多点映射」的意思就是把指定的点移动到给出的位置，从而发生形变。例如：(0, 0) -> (100, 100) 表示把 (0, 0) 位置的像素移动到 (100, 100) 的位置，这个是单点的映射，单点映射可以实现平移。而多点的映射，就可以让绘制内容任意地扭曲。

```
canvas.save();
canvas.skew(0, 0.5f);
canvas.drawBitmap(bitmap, x, y, paint);
canvas.restore();
```



原内容　　变换后

参数里，`src` 和 `dst` 是源点集合目标点集；`srcIndex` 和 `dstIndex` 是第一个点的偏移；`pointCount` 是采集的点的个数（个数不能大于 4，因为大于 4 个点就无法计算变换了）。

2.3 使用 Camera 来做三维变换

`Camera` 的三维变换有三类：旋转、平移、移动相机。

2.3.1 Camera.rotate*() 三维旋转

`Camera.rotate*()` 一共有四个方法：`rotateX(deg)` `rotateY(deg)` `rotateZ(deg)` `rotate(x, y, z)`。这四个方法的区别不用我说了吧?

```
canvas.save();

camera.rotateX(30); // 旋转 Camera 的三维空间
camera.applyToCanvas(canvas); // 把旋转投影到 Canvas

canvas.drawBitmap(bitmap, point1.x, point1.y, paint);
canvas.restore();
```

原内容

沿 x 轴旋转后

另外，`Camera` 和 `Canvas` 一样也需要保存和恢复状态才能正常绘制，不然在界面刷新之后绘制就会出现问题。所以上面这张图完整的代码应该是这样的：

```
canvas.save();

camera.save(); // 保存 Camera 的状态
camera.rotateX(30); // 旋转 Camera 的三维空间
camera.applyToCanvas(canvas); // 把旋转投影到 Canvas
camera.restore(); // 恢复 Camera 的状态

canvas.drawBitmap(bitmap, point1.x, point1.y, paint);
canvas.restore();
```

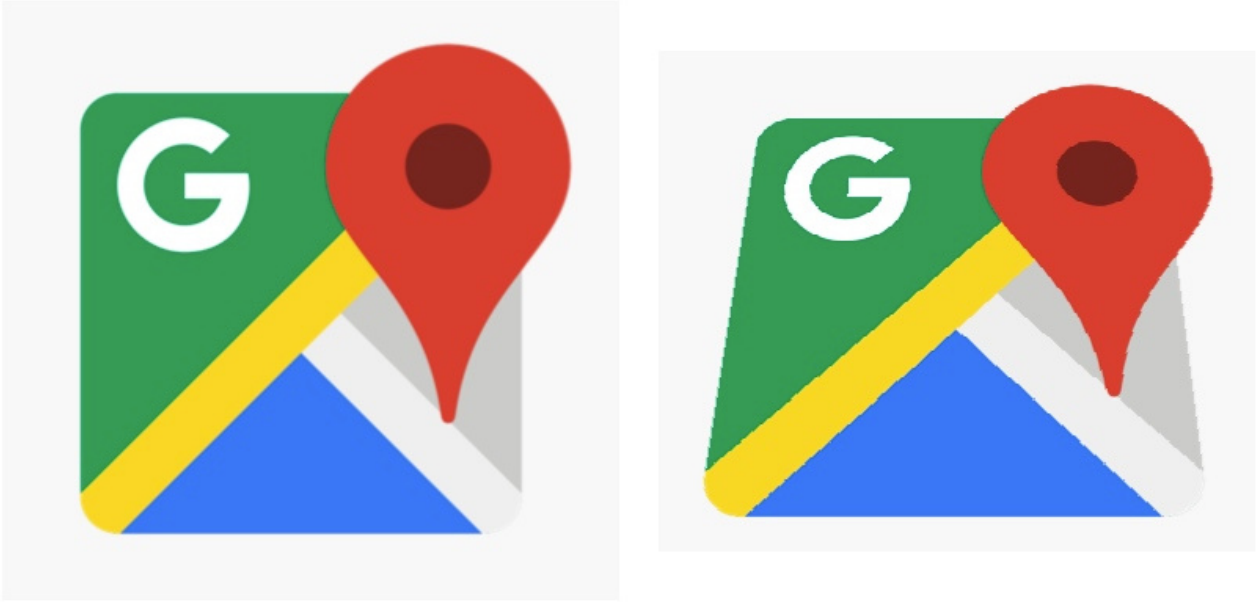
如果你需要图形左右对称，需要配合上 `Canvas.translate()`，在三维旋转之前把绘制内容的中心点移动到原点，即旋转的轴心，然后在三维旋转后再把投影移动回来：

```
canvas.save();

camera.save(); // 保存 Camera 的状态
camera.rotateX(30); // 旋转 Camera 的三维空间
canvas.translate(centerX, centerY); // 旋转之后把投影移动回来
camera.applyToCanvas(canvas); // 把旋转投影到 Canvas
canvas.translate(-centerX, -centerY); // 旋转之前把绘制内容移动到轴心（原
camera.restore(); // 恢复 Camera 的状态

canvas.drawBitmap(bitmap, point1.x, point1.y, paint);
canvas.restore();
```

> `Canvas` 的几何变换顺序是反的，所以要把移动到中心的代码写在下面，把从中心移动回来的代码写在上面。



原内容　　沿 x 轴旋转（修复旋转中心后）

2.3.2 Camera.translate(float x, float y, float z) 移动

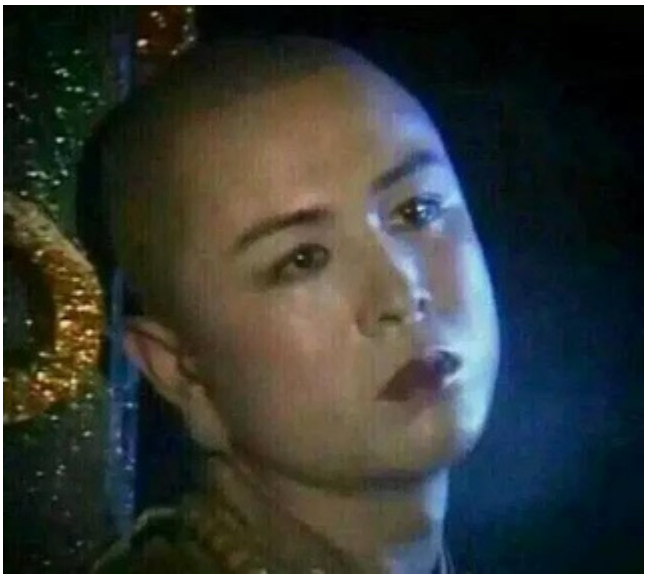
它的使用方式和 `Camera.rotate*()` 相同，而且我在项目中没有用过它，所以就不贴代码和效果图了。

2.3.3 Camera.setLocation(x, y, z) 设置虚拟相机的位置

注意！这个方法有点奇葩，它的参数的单位不是像素，而是 inch，英寸。

我 TM 的真没逗你，我也没有胡说八道，它的单位就。是。英。寸。

这种设计源自 Android 底层的图像引擎 Skia 。在 Skia 中，Camera 的位置单位是英寸，英寸和像素的换算单位在 Skia 中被写死为了 72 像素，而 Android 中把这个换算单位照搬了过来。是的，它。写。死。了。



吐槽到此为止，俗话说看透不说透，还是好朋友。

在 `Camera` 中，相机的默认位置是 (0, 0, -8)（英寸）。8 x 72 = 576，所以它的默认位置是 (0, 0, -576)（像素）。

如果绘制的内容过大，当它翻转起来的时候，就有可能出现图像投影过大的「糊脸」效果。而且由于换算单位被写死成了 72 像素，而不是和设备 dpi 相关的，所以在像素越大的手机上，这种「糊脸」效果会越明显。

而使用 `setLocation()` 方法来把相机往后移动，就可以修复这种问题。

```
camera.setLocation(0, 0, newZ);
```

`Camera.setLocation(x, y, z)` 的 `x` 和 `y` 参数一般不会改变，直接填 0 就好。

好了，上面这些就是本期的内容：范围裁切和几何变换。

练习项目

为了避免转头就忘，强烈建议你趁热打铁，做一下这个练习项目：HenCoderPracticeDraw4