自定义图标开关：
1、首先要有图片
2、自定义类继承View，重写三个构造方法
3、定义自定义属性
4、在构造中获取图片通过资源ID
　　通过样式属性得到一个样式数组
　　获取所有自定义相式
　　获取到可以滑动的最在距离的值
　　设置回收
5、测量控件的大小 onMeasure方法中
6、绘制图片的位置
7、重写onTouchEvent来设置图片滑动的位置
按下时获取按下X点的位置，滑动时获取滑动的位置-按下的位置来重新绘制滑动图片的位置
invalidate ()方法，应外了得吃，最后刷新UI
当抬起的时候来处理控件滑动到一半的问题

自定义检索条：
自定义类继承View，重写三个构造方法
1、绘制字母
准备好要画的字母，字母颜色，背景颜色
在初始化的时候创建画笔paint 设置画笔的颜色和了体的大小，去锯齿paint.setAntialias(true);
在onDraw方法中绘制字母确定字母的位置和样式drawText
默认情况下是字母的左下角的点 x:字母条宽度/2-字母宽度/2 y:单元格高度/2+字母高度/2+K
　在onMeasure方法中获取字母条的高度宽度getMeaauredwidth()以及单元格的高度　在onsizechanged()中也可以获取控件的宽高
　获取字母的大小：画笔.getTextBounds(字符，0，字符.length,rece); 表示测量整个字符串的宽高 并将其保存在rece里
2、处理字母条的触摸事件
onTouchEvent()默认的按下后会返回Flase,那么返回Flase就不能接收到后面的MOVE和UP事件 必须返回true才能接收到
当按下时通过Y轴坐标/单元格的高度，获取索引通过索引得知是按下了哪个字母
设置一个监听器将索引交给外部调用者处理
3、显示列表对列表进行分组
4、关联字母条和列表 遍历列表获得首字母然后比较也选中的是否相同，然后设置集合到指定的位置
5、显示选中的字母列表

自定义侧滑菜单：DrawerLayout
1、创建自定义父控件继承FrameLayout 原因是不用去处理一些回调方法比如说onlayout等等
2、定义子控件布局根布局就是自定义控件里面有两个子布局分别是菜单布局和内容布局
3、获取布局中的子控件通过OnfinisInflate()方法中通过getchildAt(0)去获取 布局填充后调用
4、创建ViewDragHelper对象：V4包中由2013 I/O大会提出的 用来解决控件的拖动问题
　由ViewDragHelper决定是否拦截事件，由ViewDragHelper处理触摸事件，处理回调方法
5、在构造方法中创建ViewDragHelper对象，由ViewDragHelper.create(参一，参二)来创建，参一是父控件，参二是回调callback
6、处理拦截事件交由ViewDragHelper处理
7、处理触摸事件
8、回调方法：1、捕获View 返回True就可以拖动　2、设置View的水平位置　3、设置View是否可以拖动（如果有消费事件返回大于0的数才可以拖动）决定拖动完成后动画执行的时长　4、位置发生改变的回调里通过layout方法设置子控件的位置

绘制canvas的使用

绘制直线　　通过对画笔的设置可以确定线的颜色，大小等。

canvas.drawLine(float startX, float startY, float stopX, float stopY, Paint paint)

startX：起始端点的X坐标。

startY：起始端点的Y坐标。

stopX：终止端点的X坐标。

stopY：终止端点的Y坐标。

paint：绘制直线所使用的画笔。

画布上绘制圆形

canvas.drawCircle(float cx, float cy, float radius, Paint paint);

该方法用于在画布上绘制圆形，通过指定圆形圆心的坐标和半径来实现。该方法是绘制圆形的主要方

- cx：圆心的x坐标。
- cy：圆心的y坐标。
- radius：圆的半径。
- paint：绘制时所使用的画笔。

绘制文字

canvas.drawText(String text, float x, float y, Paint paint);

x默认是text，这个字符的左边在屏幕的位置，如果设置了paint.setTextAlign(Paint.Align.CENTER)

text:绘制的文本内容

x:文本左边在屏幕上的X轴的位置

y:文本在屏幕上Y轴的位置

paint:画笔

绘制区域

drawRect(RectF rect, Paint paint)

参数一为RectF一个区域

绘制一个路径

drawPath(Path path, Paint paint)

参数一为Path路径对象

贴图

drawBitmap(Bitmap bitmap, Rect src, Rect dst, Paint paint)

参数一就是我们常规的Bitmap对象，参数二是源区域(这里是bitmap)，参数三是目标区域

画点

drawPoint(float x, float y, Paint paint)

参数一水平x轴，参数二垂直y轴，第三个参数为Paint对象。

画椭圆

drawOval(RectF oval, Paint paint)

参数一是扫描区域，参数二为paint对象；

绘制圆

drawCircle(float cx, float cy, float radius,Paint paint)

参数一是中心点的x轴，参数二是中心点的y轴，参数三是半径，参数四是paint对象；

画弧

drawArc(RectF oval, float startAngle, float sweepAngle, boolean useCenter,

参数一是RectF对象，一个矩形区域椭圆形的界限用于定义在形状、大小、电弧，参数二是

参数三扫描角(度)开始顺时针测量的，参数四是如果这是真的话,包括椭圆中心的电弧

Paint 代表了Canvas上的画笔、画刷、颜料等等；

Paint类常用方法:

setARGB(int a, int r, int g, int b) // 设置 Paint对象颜色，参数一为alpha透明值

setAlpha(int a) // 设置alpha不透明度，范围为0~255

setAntiAlias(boolean aa) // 是否抗锯齿

setColor(int color) // 设置颜色，这里Android内部定义的有Color类包含了一些常

setTextScaleX(float scaleX) // 设置文本缩放倍数，1.0f为原始

setTextSize(float textSize) // 设置字体大小

setUnderlineText(booleanunderlineText) // 设置下划线

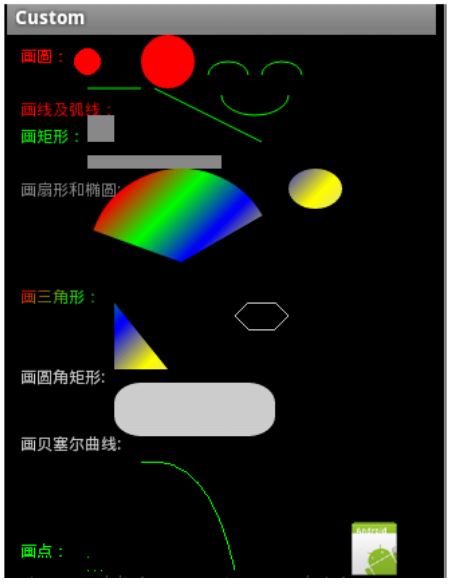
重要的类自定义View组件要重写View组件的onDraw(Canvase)方法，接下来是在该 Canvas上绘制大

```
1. public class DrawView extends View {
2.
3.     public DrawView(Context context) {
4.         super(context);
5.     }
6.
7.     @Override
8.     protected void onDraw(Canvas canvas) {
9.         super.onDraw(canvas);
```

```
10.                 /*
11.                  * 方法 说明 drawRect 绘制矩形 drawCircle 绘制圆形 drawOval 绘制椭圆 drawPath 绘制任意多边形
12.                  * drawLine 绘制直线 drawPoin 绘制点
13.                  */
14.                 // 创建画笔
15.                 Paint p = new Paint();
16.                 p.setColor(Color.RED);// 设置红色
17.
18.                 canvas.drawText("画圆：", 10, 20, p);// 画文本
19.                 canvas.drawCircle(60, 20, 10, p);// 小圆
20.                 p.setAntiAlias(true);// 设置画笔的锯齿效果。 true是去除，大家一看效果就明白了
21.                 canvas.drawCircle(120, 20, 20, p);// 大圆
22.
23.                 canvas.drawText("画线及弧线：", 10, 60, p);
24.                 p.setColor(Color.GREEN);// 设置绿色
25.                 canvas.drawLine(60, 40, 100, 40, p);// 画线
26.                 canvas.drawLine(110, 40, 190, 80, p);// 斜线
27.                 //画笑脸弧线
28.                 p.setStyle(Paint.Style.STROKE);//设置空心
29.                 RectF oval1=new RectF(150,20,180,40);
30.                 canvas.drawArc(oval1, 180, 180, false, p);//小弧形
31.                 oval1.set(190, 20, 220, 40);
32.                 canvas.drawArc(oval1, 180, 180, false, p);//小弧形
33.                 oval1.set(160, 30, 210, 60);
34.                 canvas.drawArc(oval1, 0, 180, false, p);//小弧形
35.
36.                 canvas.drawText("画矩形：", 10, 80, p);
37.                 p.setColor(Color.GRAY);// 设置灰色
38.                 p.setStyle(Paint.Style.FILL);//设置填满
39.                 canvas.drawRect(60, 60, 80, 80, p);// 正方形
40.                 canvas.drawRect(60, 90, 160, 100, p);// 长方形
41.
42.                 canvas.drawText("画扇形和椭圆:", 10, 120, p);
43.                 /* 设置渐变色 这个正方形的颜色是改变的 */
44.                 Shader mShader = new LinearGradient(0, 0, 100, 100,
45.                                 new int[] { Color.RED, Color.GREEN, Color.BLUE, Color.YELLOW,
46.                                 Color.LTGRAY }, null, Shader.TileMode.REPEAT); // 一个材质,打造出一个线性梯度沿著一条线。
47.
48.                 p.setShader(mShader);
49.                 RectF oval2 = new RectF(60, 100, 200, 240);// 设置个新的长方形，扫描测量
50.                 canvas.drawArc(oval2, 200, 130, true, p);
51.                 // 画弧，第一个参数是RectF：该类是第二个参数是角度的开始，第三个参数是多少度，第四个参数是真的时候画扇形，是假
52.                 //画椭圆，把oval改一下
53.                 oval2.set(210,100,250,130);
54.                 canvas.drawOval(oval2, p);
55.
56.                 canvas.drawText("画三角形：", 10, 200, p);
57.                 // 绘制这个三角形,你可以绘制任意多边形
58.                 Path path = new Path();
59.                 path.moveTo(80, 200);// 此点为多边形的起点
60.                 path.lineTo(120, 250);
61.                 path.lineTo(80, 250);
62.                 path.close(); // 使这些点构成封闭的多边形
63.                 canvas.drawPath(path, p);
64.
65.                 // 你可以绘制很多任意多边形，比如下面画六连形
66.                 p.reset();//重置
67.                 p.setColor(Color.LTGRAY);
68.                 p.setStyle(Paint.Style.STROKE);//设置空心
69.                 Path path1=new Path();
70.                 path1.moveTo(180, 200);
71.                 path1.lineTo(200, 200);
```

```
72.                 path1.lineTo(210,  210);
73.                 path1.lineTo(200,  220);
74.                 path1.lineTo(180,  220);
75.                 path1.lineTo(170,  210);
76.                 path1.close();//封闭
77.                 canvas.drawPath(path1,  p);
78.                 /*
79.                   *  Path类封装复合(多轮廓几何图形的路径
80.                   *  由直线段*、二次曲线,和三次方曲线，也可画以油画。drawPath(路径、油漆),要么已填充的或抚摸
81.                   *  (基于油漆的风格),或者可以用于剪断或画画的文本在路径。
82.                   */
83.
84.                 //画圆角矩形
85.                 p.setStyle(Paint.Style.FILL);//充满
86.                 p.setColor(Color.LTGRAY);
87.                 p.setAntiAlias(true);//  设置画笔的锯齿效果
88.                 canvas.drawText("画圆角矩形:",  10,  260,  p);
89.                 RectF  oval3  =  new  RectF(80,  260,  200,  300);//  设置个新的长方形
90.                 canvas.drawRoundRect(oval3,  20,  15,  p);//第二个参数是x半径，第三个参数是y半径
91.
92.                 //画贝塞尔曲线
93.                 canvas.drawText("画贝塞尔曲线:",  10,  310,  p);
94.                 p.reset();
95.                 p.setStyle(Paint.Style.STROKE);
96.                 p.setColor(Color.GREEN);
97.                 Path  path2=new  Path();
98.                 path2.moveTo(100,  320);//设置Path的起点
99.                 path2.quadTo(150,  310,  170,  400);  //设置贝塞尔曲线的控制点坐标和终点坐标
100.                canvas.drawPath(path2,  p);//画出贝塞尔曲线
101.
102.                //画点
103.                p.setStyle(Paint.Style.FILL);
104.                canvas.drawText("画点：",  10,  390,  p);
105.                canvas.drawPoint(60,  390,  p);//画一个点
106.                canvas.drawPoints(new  float[]{60,400,65,400,70,400},  p);//画多个点
107.
108.                //画图片，就是贴图
109.                Bitmap  bitmap  =  BitmapFactory.decodeResource(getResources(),  R.drawable.ic_launcher);
110.                canvas.drawBitmap(bitmap,  250,360,  p);
111.        }
112. }
```

看一下效果图：



RectF与Rect的区别：都是用于表示坐标系中的一块矩形区域，并可以对其做一些简单操作。
Rect是使用int类型作为数值，RectF是使用float类型作为数值。
两个类型提供的方法也不是完全一致。