- GestureDetectorCompat和GestureDetector是一样的,版本不同而已
- 随着手指拖动View代码

```
    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        canvas.drawBitmap(mAvatar, mOffsetX, mOffsetY, mPaint);
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        switch (event.getActionMasked()) {
            case MotionEvent.ACTION_DOWN:
                mDownX = event.getX();
                mDownY = event.getY();
                break;
            case MotionEvent.ACTION_POINTER_DOWN:
                break;
            case MotionEvent.ACTION_MOVE:
                float x = event.getX();
                float y = event.getY();
                mOffsetX += x - mDownX;
                mOffsetY += y - mDownY;
                mDownX = x;
                mDownY = y;
                invalidate();
                break;
        }
        return true;
    }
```

使用GestureDetectorCompat(手势检测器)

- 创建

```
mGestureDetectorCompat = new GestureDetectorCompat(getContext(),
this);
```

- 接管onTouchEvent

```java
@Override
public boolean onTouchEvent(MotionEvent event) {
    return mGestureDetectorCompat.onTouchEvent(event);
}
```

- 这时下面这些回调就可以响应了

```
    @Override
    public boolean onDown(MotionEvent e) {
        // 收到ActionDown事件后被调用,如果消费了返回true
        // 必须返回true,否则下面的事件都不会收到
        return false;
    }

    @Override
    public void onShowPress(MotionEvent e) {
        // 延迟后,显示按下状态时调用
    }

    @Override
    public boolean onSingleTapUp(MotionEvent e) {
        // 每次按下抬起后调用,返回值都没有用,只有onDown的返回值是有作用的
        // 长按时不会被触发了(如果长按没有开启,依然会被调用)
        // 关闭长按方法
mGestureDetectorCompat.setIsLongpressEnabled(false);
        return false;
    }

    @Override
    public boolean onScroll(MotionEvent e1, MotionEvent e2, float
distanceX, float distanceY) {
        // 手指按上去后移动了,相当于ActionMove
        return false;
    }

    @Override
    public void onLongPress(MotionEvent e) {
        // 长按GestureDetectorCompat的时间时600ms而GestureDetector
是500ms
    }

    @Override
    public boolean onFling(MotionEvent e1, MotionEvent e2, float
velocityX, float velocityY) {
        // 快速滑动,快速滑动时手指抬起了,返回值没有用
        return false;
    }
```

- 实现双击监听

```
    mGestureDetectorCompat.setOnDoubleTapListener(this);
```

```
    @Override
    public boolean onSingleTapConfirmed(MotionEvent e) {
        // 用户单击时被调用
        // 和onSingleTapUp的区别在于用户点击不会立即调用这个方法,而会在一定
时间后,确认用户没有进行双击,这个方法才会被调用
        return false;
    }

    @Override
    public boolean onDoubleTap(MotionEvent e) {
        // 用户双击时被调用
        // 注意:第二次触摸到屏幕时就调用,而不是抬起时
        if (scale == 1) {
            scale = mSmallScale;
        } else if (scale == mSmallScale) {
            scale = mBigScale;
        } else {
            scale = mSmallScale;
        }
        invalidate();
        return true;
    }

    @Override
    public boolean onDoubleTapEvent(MotionEvent e) {
        // 用户双击第二次按下时,第二次按下后移动时,第二次按下后抬起时都会被调用
        return false;
    }
```

- 如果只想实现双击监听,不实现单机可以这样

```
        mGestureDetectorCompat=new
GestureDetectorCompat(getContext(),new
GestureDetector.SimpleOnGestureListener(){
            @Override
            public boolean onDoubleTap(MotionEvent e) {
                return super.onDoubleTap(e);
            }
        });
```

- SimpleOnGestureListener中即实现了OnGestureListener,也实现了OnDoubleTapListener

OverScroller计算滑动的偏移(OverScroller并不实现动画,只作为一个计算器)

- postOnAnimation和post类似(postOnAnimation一帧执行一次,post一次会执行很多帧)
- 需要一帧一帧刷新的使用postOnAnimation
- Scroller初始速度很慢,OverScroller很快,Scroller的初始速度和没有一样,
- OverScroller的fling方法比Scroller的fling方法可以多俩个参数,表示可以过度滚动多少(滚出去多少)

```
    Paint mPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
    private static final float AVATAR_WIDTH = Utils.dp2px(200);
    private static final float OVER_SCALE_FACTORY = 1.5f;
    private Bitmap mAvatar;
    private float mOffsetX;
    private float mOffsetY;
    private float mOriginalOffsetX;
    private float mOriginalOffsetY;
    private float mSmallScale;
    private float mBigScale;
    private boolean big;
    private float scaleFraction;
    private GestureDetectorCompat mGestureDetectorCompat;
    ObjectAnimator scaleAnimator;
    private OverScroller mOverScroller;


    private float getScaleFraction() {
        return scaleFraction;
    }

    private void setScaleFraction(float scaleFraction) {
        this.scaleFraction = scaleFraction;
        invalidate();
    }

    public ScalableImageView(Context context) {
        this(context, null);
    }
    public ScalableImageView(Context context, @Nullable
AttributeSet attrs) {
        this(context, attrs, 0);
    }

    public ScalableImageView(Context context, @Nullable
AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);
        mAvatar = Utils.getAvatar(getContext(), (int)
AVATAR_WIDTH);
        // 创建检测器
        mGestureDetectorCompat = new
GestureDetectorCompat(getContext(), this);
        mGestureDetectorCompat.setOnDoubleTapListener(this);
        mGestureDetectorCompat.setIsLongpressEnabled(false);

        mOverScroller = new OverScroller(context);
    }

    private ObjectAnimator getScaleAnimator() {
        if (scaleAnimator == null) {
            scaleAnimator = ObjectAnimator.ofFloat(this,
"scaleFraction", 0, 1);
        }
        return scaleAnimator;
    }


    @Override
    protected void onSizeChanged(int w, int h, int oldw, int oldh)
{
        super.onSizeChanged(w, h, oldw, oldh);
        mOriginalOffsetX = (getWidth() - mAvatar.getWidth()) / 2f;
        mOriginalOffsetY = (getHeight() - mAvatar.getHeight()) /
2f;
        // 宽等于屏幕宽度
        mSmallScale = (float) getWidth() / mAvatar.getWidth();
        // 高等于屏幕的高度
        mBigScale = (float) getHeight() / mAvatar.getHeight() *
OVER_SCALE_FACTORY;
    }


    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        canvas.translate(mOffsetX, mOffsetY);
        float scale = mSmallScale + (mBigScale - mSmallScale) *
```

```
scaleFraction;
        canvas.scale(scale, scale, getWidth() / 2, getHeight() /
2);
        canvas.drawBitmap(mAvatar, mOriginalOffsetX,
mOriginalOffsetY, mPaint);
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        return mGestureDetectorCompat.onTouchEvent(event);
    }

    @Override
    public boolean onDown(MotionEvent e) {
        return true;
    }

    @Override
    public void onShowPress(MotionEvent e) {

    }

    @Override
    public boolean onSingleTapUp(MotionEvent e) {
        return false;
    }

    @Override
    public boolean onScroll(MotionEvent e1, MotionEvent e2, float
distanceX, float distanceY) {
        // e1按下事件
        // e2move事件
        // distanceX，distanceY上一个点到当前点的距离
        if (big) {
            mOffsetX -= distanceX;
            mOffsetY -= distanceY;
            mOffsetX = Math.min(mOffsetX, (mAvatar.getWidth() *
mBigScale - getWidth()) / 2);
            mOffsetX = Math.max(mOffsetX, -(mAvatar.getWidth() *
mBigScale - getWidth()) / 2);
            mOffsetY -= distanceY;
            mOffsetY = Math.min(mOffsetY, (mAvatar.getHeight() *
mBigScale - getHeight()) / 2);
            mOffsetY = Math.max(mOffsetY, -(mAvatar.getHeight() *
mBigScale - getHeight()) / 2);
            invalidate();
        }
        return false;
    }

    @Override
    public void onLongPress(MotionEvent e) {

    }

    @Override
    public boolean onFling(MotionEvent e1, MotionEvent e2, float
velocityX, float velocityY) {
        if (big) {
            /**
             * mOffsetX,mOffsetY初始的位置
             * velocityX, velocityY, x, y方向上的速度
             * minX,maxX,minY,maxY
             * 把中间定为圆点会更好算
             * int minX, int maxX, int minY, int maxY代表圆点可以移动的
最大的区域(那个模型)
             */
            mOverScroller.fling((int) mOffsetX, (int) mOffsetY,
(int) velocityX, (int) velocityY,
                    -(int) (mAvatar.getWidth() * mBigScale -
getWidth()) / 2,
                    (int) (mAvatar.getWidth() * mBigScale -
getWidth()) / 2,
                    -(int) (mAvatar.getHeight() * mBigScale -
getHeight()) / 2,
                    (int) (mAvatar.getHeight() * mBigScale -
getHeight()) / 2);
            // 作用是让runnable中的代码在下一帧执行
            postOnAnimation(this);
        }
        return false;
    }

    @Override
    public void run() {
        if (mOverScroller.computeScrollOffset()) {//动画是否还在执行中
            mOffsetX = mOverScroller.getCurrX();
            mOffsetY = mOverScroller.getCurrY();
            invalidate();
            // 每一帧都自动执行下一帧
            postOnAnimation(this);
        }
    }

    @Override
    public boolean onSingleTapConfirmed(MotionEvent e) {
        return false;
    }

    @Override
    public boolean onDoubleTap(MotionEvent e) {
        big = !big;
        if (big) {
            getScaleAnimator().start();
        } else {
            getScaleAnimator().reverse();
        }
        return true;
    }

    @Override
    public boolean onDoubleTapEvent(MotionEvent e) {
        return false;
    }
```