11.动画

https://blog.csdn.net/carson_ho/article/details/79860980

https://anriku.top/2018/09/01/Android%E5%B1%9E%E6%80%A7%E5%8A%A8%E7%94%BB%E6%8E%A2%E7%B4%A2/

https://blog.csdn.net/carson_ho/article/details/72827747

https://juejin.cn/post/6846687601118691341#heading-12

11.1. 动画的类型

11.1.1.视图动画

https://blog.csdn.net/carson_ho/article/details/72827747 // 以下参数是4种动画效果的公共属性,即都有的属性 android:duration="3000" // 动画持续时间（ms），必须设置，动画才有效果 android:startOffset ="1000" // 动画延迟开始时间（ms） android:fillBefore = “true” // 动画播放完后，视图是否会停留在动画开始的状态，默认为true android:fillAfter = “false” // 动画播放完后，视图是否会停留在动画结束的状态，优先于fillBefore值，默认为false android:fillEnabled= “true” // 是否应用fillBefore值，对fillAfter值无影响，默认为true android:repeatMode= “restart” // 选择重复播放动画模式，restart代表正序重放，reverse代表倒序回放，默认为restart| android:repeatCount = “0” // 重放次数（所以动画的播放次数=重放次数+1），为infinite时无限重复 android:interpolator = @nim/interpolator_resource // 插值器，即影响动画的播放速度,下面会详细讲

11.1.2补间动画

平移动画（Translate） // 1. fromXDelta ：视图在水平方向x 移动的起始值 // 2. toXDelta ：视图在水平方向x 移动的结束值 // 3. fromYDelta ：视图在竖直方向y 移动的起始值 // 4. toYDelta：视图在竖直方向y 移动的结束值 缩放动画（scale） android:fromXScale="0.0" // 动画在水平方向X的起始缩放倍数 // 0.0表示收缩到没有；1.0表示正常无伸缩 // 值小于1.0表示收缩；值大于1.0表示放大 android:toXScale="2" //动画在水平方向X的结束缩放倍数 android:fromYScale="0.0" //动画开始前在竖直方向Y的起始缩放倍数 android:toYScale="2" //动画在竖直方向Y的结束缩放倍数 android:pivotX="50%" // 缩放轴点的x坐标 android:pivotY="50%" // 缩放轴点的y坐标 旋转动画（rotate） android:fromDegrees="0" // 动画开始时 视图的旋转角度(正数 = 顺时针，负数 = 逆时针) android:toDegrees="270" // 动画结束时 视图的旋转角度(正数 = 顺时针，负数 = 逆时针) android:pivotX="50%" // 旋转轴点的x坐标 android:pivotY="0" // 旋转轴点的y坐标 透明度动画（alpha） android:fromAlpha="1.0" // 动画开始时视图的透明度(取值范围: -1 ~ 1) android:toAlpha="0.0"// 动画结束时视图的透明度(取值范围: -1 ~ 1)

11.1.3逐帧动画

https://blog.csdn.net/carson_ho/article/details/73087488

按序播放一组预先定义好的图片

xml:animation-list

// item = 动画图片资源；duration = 设置一帧持续时间(ms)

属性动画

what

顾名思义，通过控制对象的属性，来实现动画效果。官方定义：定义一个随着时间（注：停个顿）更改任何对象属性的动画，无论其是否绘制到屏幕上

链接：https://www.jianshu.com/p/821ef6d1e1c9

Duration：定义动画时长，默认是300 ms。

Time interpolation: 时间插值器，它可以指定属性值如何随时间变化的，反应了动画的运动速率。

Repeat count and behavior: 指定当动画结束时是否重复动画以及动画重复多少次，还可以设置反向播放动画，播放到达指定次数后动画结束。

Animator sets：把一组动画聚在一起，顺序播放或者同时播放或者延迟播放。

Frame refresh delay：指定刷新动画帧的频率，默认时间是10ms，但是刷新频率最终取决于系统是否繁忙以及系统服务底层计时器的快慢。

why

11.2. 补间动画和属性动画的区别

a. 作用对象局限：View

补间动画 只能够作用在视图View上，即只可以对一个Button、TextView、甚至是LinearLayout、或者其它继承自View的组件进行动画操作，但无法对非View的对象进行动画操。

有些情况下的动画效果只是视图的某个属性 & 对象而不是整个视图； 如，现需要实现视图的颜色动态变化，那么就需要操作视图的颜色属性从而实现动画效果，而不是针对整个视图进行动画操作

b. 没有改变View的属性，只是改变视觉效果

补间动画只是改变了View的视觉效果，而不会真正去改变View的属性。

如，将屏幕左上角的按钮 通过补间动画 移动到屏幕的右下角

点击当前按钮位置（屏幕右下角）是没有效果的，因为实际上按钮还是停留在屏幕左上角，补间动画只是将这个按钮绘制到屏幕右下角，改变了视觉效果而已。

c. 动画效果单一

补间动画只能实现平移、旋转、缩放 & 透明度这些简单的动画需求

11.3. ObjectAnimator，ValueAnimator及其区别

ObjectAnimator

位移

val objectAnimation =ObjectAnimator.ofFloat(llAddAccount, "translationX", 0f, -70f) 第三参数为可变长参数，第一个值为动画开始的位置，第二个值为结束值得位置，如果数组大于3位数，那么前者将是后者的起始位置

旋转 val objectAnimation = ObjectAnimator.ofFloat(tvText, "rotation", 0f,180f,0f)

```
    ofFloat()方法的可变长参数，如果后者的值大于前者，那么顺时针旋转，小于前者，则逆时针旋转。
```

缩放

```
val objectAnimation =ObjectAnimator.ofFloat(tvText, "scaleX", 1f,2f)
```

ofFloat()方法传入参数属性为scaleX和scaleY时，动态参数表示缩放的倍数

透明 val objectAnimation =ObjectAnimator.ofFloat(tvText, "alpha", 1f,0f,1f)

ValueAnimator

val valueAnimator = ValueAnimator.ofFloat(0f, 180f) valueAnimator.addUpdateListener { tvText.rotationY = it.animatedValue as Float //手动赋值 } valueAnimator.start()

等价于

```
ObjectAnimator.ofFloat(tvText, "rotationY", 0f, 180f).apply { start() }
```

ValueAnimator作为ObjectAnimator的父类，主要动态计算目标对象属性的值，然后设置给对象属性，达到动画效果

使用ValueAnimator实现动画，需要手动赋值给目标对象tvText的rotationY，而ObjectAnimator则是自动赋值，不需要手动赋值就可以达到效果

ObjectAnimator和ValueAnimator区别

其实二者都是属于属性动画，本质上是一样的，都是先改 变值，然后赋值给对象属性，从而实现动画操作。

```
但二者区别就在与，ValueAnimator类是　手动　赋值给对象的属性，从而实现动画，
```

而ObjectAnimator类，是 自动 赋值给对象的属性，从

AnimatorSet

一个动画结束后播放另外一个动画，或者同时播放

val aAnimator=ObjectAnimator.ofInt(1) val bAnimator=ObjectAnimator.ofInt(1) val cAnimator=ObjectAnimator.ofInt(1) val dAnimator=ObjectAnimator.ofInt(1)

AnimatorSet().apply { play(aAnimator).before(bAnimator)//a 在b之前播放 play(bAnimator).with(cAnimator)//b和c同时播放动画效果 play(dAnimator).after(cAnimator)//d 在c播放结束之后播放 start() }

11.4. TimeInterpolator插值器，自定义插值器

TimeInterpolator

它的作用是根据时间流逝的百分比来计算出当前属性值改变的百分比

插值器 ，Interpolator负责控制动画变化的速率，使得基本的动画效果能够以匀速、加速、减速、抛物线速率等各种速率变化

匀速插值器 https://blog.csdn.net/qinxiandiqi/article/details/51719926 图1假设了一个对象需要对它的x属性设定动画，这个x属性代表这个对象在屏幕上面的横坐标。这个动画的时间设定为40毫秒，以及需要移动的距离是40个像素。每过10毫秒（默认的帧频率），这个对象就会在水平方向上移动10个像素。等到40毫秒之后，这个动画停止，对象在水平方向上总共移动了40个像素。这个例子中的动画使用了一个线性插值器，意味着这个对象以匀速移动。

例子: http://static.kancloud.cn/alex_wsc/android_art/1828610

，当时间t=20ms的时候，时间流逝的百分比是0.5（20/40=0.5）意味着现在时间过了一半，那x应该改变多少呢这个就由插值器和估值算法来确定

拿线性插值器来说，当时间流逝一半的时候，x的变换也应该是一半，即x的改变是0.5，为什么呢？因为它是线性插值器，是实现匀速动画的

估值器evaluate的三个参数分别表示估值小数、开始值和结束值，对应于我们的例子就分别是0.5、0、40。根据上述算法，整型估值返回给我们的结果是20，这就是（x=20, t=20ms）的由来。

系统已有的插值器： ①LinearInterpolator（线性插值器）：匀速动画。 ②AccelerateDecelerateInterpolator（加速减速插值器）：动画两头慢，中间快。 ③DecelerateInterpolator（减速插值器）：动画越来越慢。

自定义插值器

写一个自定义Interpolator：先减速后加速

public class DecelerateAccelerateInterpolator implements TimeInterpolator {

```
@Override
public float getInterpolation(float input) {
    float result;
    if (input <= 0.5) {
        result = (float) (Math.sin(Math.PI * input)) / 2;
        // 使用正弦函数来实现先减速后加速的功能，逻辑如下：
        // 因为正弦函数初始弧度变化值非常大，刚好和余弦函数是相反的
        // 随着弧度的增加，正弦函数的变化值也会逐渐变小，这样也就实现了减速的效果。
        // 当弧度大于π/2之后，整个过程相反了过来，现在正弦函数的弧度变化值非常小，渐渐随着弧度继续增加，变化值越来越大，弧度到π时结束，这样从0过度到π，也就实现了先减速后加速的效果
    } else {
        result = (float) (2 - Math.sin(Math.PI * input)) / 2;
    }
    return result;
    // 返回的result值 = 随着动画进度呈先减速后加速的变化趋势
}
```

https://blog.csdn.net/carson_ho/article/details/72863901

11.5. TypeEvaluator估值器

TypeEvaluator

估值器 据当前属性改变的百分比来计算改变后的属性值

①IntEvaluator：针对整型属性 ②FloatEvaluator：针对浮点型属性 ③ArgbEvaluator：针对Color属性

public class FloatEvaluator implements TypeEvaluator { // FloatEvaluator实现了TypeEvaluator接口

// 重写evaluate() public Object evaluate(float fraction, Object startValue, Object endValue) { // 参数说明 // fraction：表示动画完成度（根据它来计算当前动画的值） // startValue、endValue：动画的初始值和结束值 float startFloat = ((Number) startValue).floatValue();

```
    return startFloat + fraction * (((Number) endValue).floatValue() - startFloat);
    // 初始值 过渡 到结束值 的算法是：
    // 1\. 用结束值减去初始值，算出它们之间的差值
    // 2\. 用上述差值乘以fraction系数
    // 3\. 再加上初始值，就得到当前动画的值
}
```

实例

实现的动画效果：一个圆从一个点 移动到 另外一个点

https://blog.csdn.net/carson_ho/article/details/72863901

1.onDraw画圆canvas.drawCircle(x, y, RADIUS, mPaint)

2:创建动画对象 & 设置初始值 和 结束值 ValueAnimator anim = ValueAnimator.ofObject(new PointEvaluator(), startPoint, endPoint);

3.自定义PointEvaluator,evaluate根据插值器的速率计算出Point 的x,y 4.通过 值 的更新监听器，将改变的对象手动赋值给当前对象

5.每次赋值后就重新绘制，从而实现动画效果