**[安卓开发之实现控件的贝塞尔曲线路径动画](http://blog.csdn.net/cxmscb/article/details/52485878)**

贝塞尔曲线一般用在绘制canvas图层上，这里主要通过ObjectAnimation动画及估值器实现将贝塞尔曲线作为控件移动的动画路径。

## 一、ObjectAnimator动画原理

ObjectAnimator.ofObject(….,”xxx”,估值值,区间数组); 【定义动画属性xxx和区间】

插值器/加速器（Interpolator）【返回当前数字进度t】

估值值（Evaluator）【根当前数字进度计算并返回当前值】

调用setXxx函数 【根据封装好的setXxx函数并反射调用，将第三步返回当前值以参数传入】

## 二、自定义ObjectAnimator属性

上面我们理解了ObjectAnimator的动画实现原理，下面我们来自定义一个属性动画来看看实现贝塞尔路径效果吧。

### 一、保存贝塞尔路径的信息——ViewPoint

为了保存贝塞尔路径的一些关键点信息，我们定义一个类来保存：

public class ViewPoint{

float x ,y;

float x1,y1;

float x2,y2;

int operation;

public ViewPoint(float x, float y) {

this.x = x;

this.y = y;

}

// 设置起点

public static ViewPoint moveTo(float x, float y, int operation){

return new ViewPoint(x,y,operation);

}

// 直线移动

public static ViewPoint lineTo(float x, float y, int operation){

return new ViewPoint(x,y,operation);

}

// 三阶贝塞尔曲线移动

public static ViewPoint curveTo(float x, float y,float x1,float y1,float x2,float y2, int operation){

return new ViewPoint(x,y,x1,y1,x2,y2,operation);

}

// 二阶贝塞尔曲线移动

public static ViewPoint quadTo(float x, float y,float x1,float y1, int operation){

return new ViewPoint(x,y,x1,y1,operation);

}

private ViewPoint(float x, float y, int operation) {

this.x = x;

this.y = y;

this.operation = operation;

}

public ViewPoint(float x, float y, float x1, float y1, int operation) {

this.x = x;

this.y = y;

this.x1 = x1;

this.y1 = y1;

this.operation = operation;

}

public ViewPoint(float x, float y, float x1, float y1, float x2, float y2, int operation) {

this.x = x;

this.y = y;

this.x1 = x1;

this.y1 = y1;

this.x2 = x2;

this.y2 = y2;

this.operation = operation;

}

}

### 二、保存贝塞尔路径的集合——ViewPath

为了实现跟绘图时连续画线的效果，我们需要一个路径信息Path将路径的操作先后保存起来：

import java.util.ArrayList;

import java.util.Collection;

/\*\*

\* Created by cxm on 2016/9/8.

\*/

public class ViewPath {

// 四种路径操作

public static final int MOVE = 0;

public static final int LINE = 1;

public static final int QUAD = 2;

public static final int CURVE = 3;

private ArrayList<ViewPoint> mPoints;

public ViewPath() {

mPoints = new ArrayList<>();

}

// 添加重置起点的路径

public void moveTo(float x, float y){

mPoints.add(ViewPoint.moveTo(x,y,MOVE));

}

// 添加直线移动的路径

public void lineTo(float x,float y){

mPoints.add(ViewPoint.lineTo(x,y,LINE));

}

// 添加三阶贝塞尔移动的路径

public void curveTo(float x,float y,float x1,float y1,float x2,float y2){

mPoints.add(ViewPoint.curveTo(x,y,x1,y1,x2,y2,CURVE));

}

// 添加二阶贝塞尔移动的路径

public void quadTo(float x,float y,float x1,float y1){

mPoints.add(ViewPoint.quadTo(x,y,x1,y1,QUAD));

}

// 获取路径集合

public Collection<ViewPoint> getPoints(){

return mPoints;

}

}

### 三、自定义ObjectAnimation动画：

ViewPath path = new ViewPath(); //保存View的移动路径

path.moveTo(0,0);

path.lineTo(0,500);

path.curveTo(-300,200,-600,800,-800,400);

path.lineTo(-800,100);

// 自定义fabLoc动画。

// 第三个参数传入用于计算坐标的估值器，第四个参数传入用于估值的路径集合

ObjectAnimator anim = ObjectAnimator.ofObject(this,"fabLoc",new ViewPathEvaluator(),path.getPoints().toArray());

anim.setInterpolator(new AccelerateDecelerateInterpolator());

anim.setDuration(3000);

anim.start();

// 基于ObjectAnimation的实现原理定义：定义setFabLoc函数。参数为路径信息对象

public void setFabLoc(ViewPoint newLoc){

imageButton.setTranslationX(newLoc.x);

imageButton.setTranslationY(newLoc.y);

}

### 四、用于计算路径坐标的估值器ViewPathEvaluator

在自定义的估值器中，我们需要根据前后两个路径信息来计算应该返回的偏移坐标x，y。计算时我们需要用到贝塞尔曲线的路径公式。

公式可参考链接: [贝塞尔曲线介绍与公式](http://www.2cto.com/kf/201604/497130.html)

import android.animation.TypeEvaluator;

/\*\*

\* Created by cxm on 2016/9/8.

\*/

public class ViewPathEvaluator implements TypeEvaluator<ViewPoint> {

// 自定义估值器：ViewPathEvaluator

public ViewPathEvaluator() {

}

@Override

public ViewPoint evaluate(float t, ViewPoint startValue, ViewPoint endValue) {

// 返回一个路径信息，其中包含偏移坐标x和偏移坐标y。

// 【返回用于反射调用setFabLoc时函数的传参】、

// startValue:前一个操作路径 endValue：后一个操作路径 t：操作进度(0->1)

// startValue和endValue为传入的路径集合数组中相邻的两个路径

float x ,y;

float startX,startY;

//分情况进行判断，计算出返回的偏移坐标：

if(endValue.operation == ViewPath.LINE){

//起点判断：

startX = (startValue.operation==ViewPath.QUAD)?startValue.x1:startValue.x;

startX = (startValue.operation == ViewPath.CURVE)?startValue.x2:startX;

startY = (startValue.operation==ViewPath.QUAD)?startValue.y1:startValue.y;

startY = (startValue.operation == ViewPath.CURVE)?startValue.y2:startY;

//返回的偏移坐标计算：根据公式

x = startX + t \* (endValue.x - startX);

y = startY+ t \* (endValue.y - startY);

}else if(endValue.operation == ViewPath.CURVE){

//起点判断：

startX = (startValue.operation==ViewPath.QUAD)?startValue.x1:startValue.x;

startY = (startValue.operation==ViewPath.QUAD)?startValue.y1:startValue.y;

float oneMinusT = 1 - t;

//返回的偏移坐标计算：根据公式

x = oneMinusT \* oneMinusT \* oneMinusT \* startX +

3 \* oneMinusT \* oneMinusT \* t \* endValue.x +

3 \* oneMinusT \* t \* t \* endValue.x1+

t \* t \* t \* endValue.x2;

y = oneMinusT \* oneMinusT \* oneMinusT \* startY +

3 \* oneMinusT \* oneMinusT \* t \* endValue.y +

3 \* oneMinusT \* t \* t \* endValue.y1+

t \* t \* t \* endValue.y2;

}else if(endValue.operation == ViewPath.MOVE){

x = endValue.x;

y = endValue.y;

}else if(endValue.operation == ViewPath.QUAD){

//起点判断：

startX = (startValue.operation==ViewPath.CURVE)?startValue.x2:startValue.x;

startY = (startValue.operation==ViewPath.CURVE)?startValue.y2:startValue.y;

//返回的偏移坐标计算：根据公式

float oneMinusT = 1 - t;

x = oneMinusT \* oneMinusT \* startX +

2 \* oneMinusT \* t \* endValue.x +

t \* t \* endValue.x1;

y = oneMinusT \* oneMinusT \* startY +

2 \* oneMinusT \* t \* endValue.y +

t \* t \* endValue.y1;

}else {

// 其他

x = endValue.x;

y = endValue.y;

}

return new ViewPoint(x,y);

}

}

返回偏移坐标x和y后，系统就会不断地反射调用setFabLoc函数让控件偏移：

// 基于ObjectAnimation的实现原理定义：定义setFabLoc函数。参数为路径信息对象

public void setFabLoc(ViewPoint newLoc){

imageButton.setTranslationX(newLoc.x);

imageButton.setTranslationY(newLoc.y);

}

### 后续：

1. 因ObjectAnimation中的反射函数位置的原因，无法进行进一步的动画封装。