转载请注明出处：<http://blog.csdn.net/guolin_blog/article/details/43536355>

大家好，在上一篇文章当中，我们学习了[**Android**](http://lib.csdn.net/base/15)属性动画的基本用法，当然也是最常用的一些用法，这些用法足以覆盖我们平时大多情况下的动画需求了。但是，正如上篇文章当中所说到的，属性动画对补间动画进行了很大幅度的改进，之前补间动画可以做到的属性动画也能做到，补间动画做不到的现在属性动画也可以做到了。因此，今天我们就来学习一下属性动画的高级用法，看看如何实现一些补间动画所无法实现的功能。

阅读本篇文章需要你对属性动画有一定的了解，并且掌握属性动画的基本用法，如果你还对属性动画不够了解的话，建议先去阅读 [**Android属性动画完全解析(上)，初识属性动画的基本用法**](http://blog.csdn.net/guolin_blog/article/details/43536355) 。

**ValueAnimator的高级用法**

在上篇文章中介绍补间动画缺点的时候有提到过，补间动画是只能对View对象进行动画操作的。而属性动画就不再受这个限制，它可以对任意对象进行动画操作。那么大家应该还记得在上篇文章当中我举的一个例子，比如说我们有一个自定义的View，在这个View当中有一个Point对象用于管理坐标，然后在onDraw()方法当中就是根据这个Point对象的坐标值来进行绘制的。也就是说，如果我们可以对Point对象进行动画操作，那么整个自定义View的动画效果就有了。OK，下面我们就来学习一下如何实现这样的效果。

在开始动手之前，我们还需要掌握另外一个知识点，就是TypeEvaluator的用法。可能在大多数情况下我们使用属性动画的时候都不会用到TypeEvaluator，但是大家还是应该了解一下它的用法，以防止当我们遇到一些解决不掉的问题时能够想起来还有这样的一种解决方案。

那么TypeEvaluator的作用到底是什么呢？简单来说，就是告诉动画系统如何从初始值过度到结束值。我们在上一篇文章中学到的ValueAnimator.ofFloat()方法就是实现了初始值与结束值之间的平滑过度，那么这个平滑过度是怎么做到的呢？其实就是系统内置了一个FloatEvaluator，它通过计算告知动画系统如何从初始值过度到结束值，我们来看一下FloatEvaluator的代码实现：

**[java]** [view plain](http://blog.csdn.net/guolin_blog/article/details/43816093) [copy](http://blog.csdn.net/guolin_blog/article/details/43816093)

[在CODE上查看代码片](https://code.csdn.net/snippets/660782)

1. **public** **class** FloatEvaluator **implements** TypeEvaluator {
2. **public** Object evaluate(**float** fraction, Object startValue, Object endValue) {
3. **float** startFloat = ((Number) startValue).floatValue();
4. **return** startFloat + fraction \* (((Number) endValue).floatValue() - startFloat);
5. }
6. }

可以看到，FloatEvaluator实现了TypeEvaluator接口，然后重写evaluate()方法。evaluate()方法当中传入了三个参数，第一个参数fraction非常重要，这个参数用于表示动画的完成度的，我们应该根据它来计算当前动画的值应该是多少，第二第三个参数分别表示动画的初始值和结束值。那么上述代码的逻辑就比较清晰了，用结束值减去初始值，算出它们之间的差值，然后乘以fraction这个系数，再加上初始值，那么就得到当前动画的值了。

好的，那FloatEvaluator是系统内置好的功能，并不需要我们自己去编写，但介绍它的实现方法是要为我们后面的功能铺路的。前面我们使用过了ValueAnimator的ofFloat()和ofInt()方法，分别用于对浮点型和整型的数据进行动画操作的，但实际上ValueAnimator中还有一个ofObject()方法，是用于对任意对象进行动画操作的。但是相比于浮点型或整型数据，对象的动画操作明显要更复杂一些，因为系统将完全无法知道如何从初始对象过度到结束对象，因此这个时候我们就需要实现一个自己的TypeEvaluator来告知系统如何进行过度。

下面来先定义一个Point类，如下所示：

**[java]** [view plain](http://blog.csdn.net/guolin_blog/article/details/43816093) [copy](http://blog.csdn.net/guolin_blog/article/details/43816093)

[在CODE上查看代码片](https://code.csdn.net/snippets/660782)

1. **public** **class** Point {
3. **private** **float** x;
5. **private** **float** y;
7. **public** Point(**float** x, **float** y) {
8. **this**.x = x;
9. **this**.y = y;
10. }
12. **public** **float** getX() {
13. **return** x;
14. }
16. **public** **float** getY() {
17. **return** y;
18. }
20. }

Point类非常简单，只有x和y两个变量用于记录坐标的位置，并提供了构造方法来设置坐标，以及get方法来获取坐标。接下来定义PointEvaluator，如下所示：

**[java]** [view plain](http://blog.csdn.net/guolin_blog/article/details/43816093) [copy](http://blog.csdn.net/guolin_blog/article/details/43816093)

[在CODE上查看代码片](https://code.csdn.net/snippets/660782)

1. **public** **class** PointEvaluator **implements** TypeEvaluator{
3. @Override
4. **public** Object evaluate(**float** fraction, Object startValue, Object endValue) {
5. Point startPoint = (Point) startValue;
6. Point endPoint = (Point) endValue;
7. **float** x = startPoint.getX() + fraction \* (endPoint.getX() - startPoint.getX());
8. **float** y = startPoint.getY() + fraction \* (endPoint.getY() - startPoint.getY());
9. Point point = **new** Point(x, y);
10. **return** point;
11. }
13. }

可以看到，PointEvaluator同样实现了TypeEvaluator接口并重写了evaluate()方法。其实evaluate()方法中的逻辑还是非常简单的，先是将startValue和endValue强转成Point对象，然后同样根据fraction来计算当前动画的x和y的值，最后组装到一个新的Point对象当中并返回。

这样我们就将PointEvaluator编写完成了，接下来我们就可以非常轻松地对Point对象进行动画操作了，比如说我们有两个Point对象，现在需要将Point1通过动画平滑过度到Point2，就可以这样写：

**[java]** [view plain](http://blog.csdn.net/guolin_blog/article/details/43816093) [copy](http://blog.csdn.net/guolin_blog/article/details/43816093)

[在CODE上查看代码片](https://code.csdn.net/snippets/660782)

1. Point point1 = **new** Point(0, 0);
2. Point point2 = **new** Point(300, 300);
3. ValueAnimator anim = ValueAnimator.ofObject(**new** PointEvaluator(), point1, point2);
4. anim.setDuration(5000);
5. anim.start();

代码很简单，这里我们先是new出了两个Point对象，并在构造函数中分别设置了它们的坐标点。然后调用ValueAnimator的ofObject()方法来构建ValueAnimator的实例，这里需要注意的是，ofObject()方法要求多传入一个TypeEvaluator参数，这里我们只需要传入刚才定义好的PointEvaluator的实例就可以了。

好的，这就是自定义TypeEvaluator的全部用法，掌握了这些知识之后，我们就可以来尝试一下如何通过对Point对象进行动画操作，从而实现整个自定义View的动画效果。

新建一个MyAnimView继承自View，代码如下所示：

**[java]** [view plain](http://blog.csdn.net/guolin_blog/article/details/43816093) [copy](http://blog.csdn.net/guolin_blog/article/details/43816093)

[在CODE上查看代码片](https://code.csdn.net/snippets/660782)

1. **public** **class** MyAnimView **extends** View {
3. **public** **static** **final** **float** RADIUS = 50f;
5. **private** Point currentPoint;
7. **private** Paint mPaint;
9. **public** MyAnimView(Context context, AttributeSet attrs) {
10. **super**(context, attrs);
11. mPaint = **new** Paint(Paint.ANTI\_ALIAS\_FLAG);
12. mPaint.setColor(Color.BLUE);
13. }
15. @Override
16. **protected** **void** onDraw(Canvas canvas) {
17. **if** (currentPoint == **null**) {
18. currentPoint = **new** Point(RADIUS, RADIUS);
19. drawCircle(canvas);
20. startAnimation();
21. } **else** {
22. drawCircle(canvas);
23. }
24. }
26. **private** **void** drawCircle(Canvas canvas) {
27. **float** x = currentPoint.getX();
28. **float** y = currentPoint.getY();
29. canvas.drawCircle(x, y, RADIUS, mPaint);
30. }
32. **private** **void** startAnimation() {
33. Point startPoint = **new** Point(RADIUS, RADIUS);
34. Point endPoint = **new** Point(getWidth() - RADIUS, getHeight() - RADIUS);
35. ValueAnimator anim = ValueAnimator.ofObject(**new** PointEvaluator(), startPoint, endPoint);
36. anim.addUpdateListener(**new** ValueAnimator.AnimatorUpdateListener() {
37. @Override
38. **public** **void** onAnimationUpdate(ValueAnimator animation) {
39. currentPoint = (Point) animation.getAnimatedValue();
40. invalidate();
41. }
42. });
43. anim.setDuration(5000);
44. anim.start();
45. }
47. }

基本上还是很简单的，总共也没几行代码。首先在自定义View的构造方法当中初始化了一个Paint对象作为画笔，并将画笔颜色设置为蓝色，接着在onDraw()方法当中进行绘制。这里我们绘制的逻辑是由currentPoint这个对象控制的，如果currentPoint对象不等于空，那么就调用drawCircle()方法在currentPoint的坐标位置画出一个半径为50的圆，如果currentPoint对象是空，那么就调用startAnimation()方法来启动动画。

那么我们来观察一下startAnimation()方法中的代码，其实大家应该很熟悉了，就是对Point对象进行了一个动画操作而已。这里我们定义了一个startPoint和一个endPoint，坐标分别是View的左上角和右下角，并将动画的时长设为5秒。然后有一点需要大家注意的，就是我们通过监听器对动画的过程进行了监听，每当Point值有改变的时候都会回调onAnimationUpdate()方法。在这个方法当中，我们对currentPoint对象进行了重新赋值，并调用了invalidate()方法，这样的话onDraw()方法就会重新调用，并且由于currentPoint对象的坐标已经改变了，那么绘制的位置也会改变，于是一个平移的动画效果也就实现了。

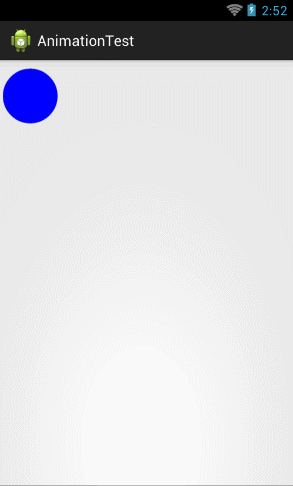
下面我们只需要在布局文件当中引入这个自定义控件：

**[html]** [view plain](http://blog.csdn.net/guolin_blog/article/details/43816093) [copy](http://blog.csdn.net/guolin_blog/article/details/43816093)

[在CODE上查看代码片](https://code.csdn.net/snippets/660782)

1. **<RelativeLayout** xmlns:android="http://schemas.android.com/apk/res/android"
2. android:layout\_width="match\_parent"
3. android:layout\_height="match\_parent"
4. **>**
6. **<com.example.tony.myapplication.MyAnimView**
7. android:layout\_width="match\_parent"
8. android:layout\_height="match\_parent" **/>**
10. **</RelativeLayout>**

最后运行一下程序，效果如下图所示：



OK！这样我们就成功实现了通过对对象进行值操作来实现动画效果的功能，这就是ValueAnimator的高级用法。

**ObjectAnimator的高级用法**

ObjectAnimator的基本用法和工作原理在上一篇文章当中都已经讲解过了，相信大家都已经掌握。那么大家应该都还记得，我们在吐槽补间动画的时候有提到过，补间动画是只能实现移动、缩放、旋转和淡入淡出这四种动画操作的，功能限定死就是这些，基本上没有任何扩展性可言。比如我们想要实现对View的颜色进行动态改变，补间动画是没有办法做到的。

但是属性动画就不会受这些条条框框的限制，它的扩展性非常强，对于动态改变View的颜色这种功能是完全可是胜任的，那么下面我们就来学习一下如何实现这样的效果。

大家应该都还记得，ObjectAnimator内部的工作机制是通过寻找特定属性的get和set方法，然后通过方法不断地对值进行改变，从而实现动画效果的。因此我们就需要在MyAnimView中定义一个color属性，并提供它的get和set方法。这里我们可以将color属性设置为字符串类型，使用#RRGGBB这种格式来表示颜色值，代码如下所示：

**[java]** [view plain](http://blog.csdn.net/guolin_blog/article/details/43816093) [copy](http://blog.csdn.net/guolin_blog/article/details/43816093)

[在CODE上查看代码片](https://code.csdn.net/snippets/660782)

1. **public** **class** MyAnimView **extends** View {
3. ...
5. **private** String color;
7. **public** String getColor() {
8. **return** color;
9. }
11. **public** **void** setColor(String color) {
12. **this**.color = color;
13. mPaint.setColor(Color.parseColor(color));
14. invalidate();
15. }
17. ...
19. }

注意在setColor()方法当中，我们编写了一个非常简单的逻辑，就是将画笔的颜色设置成方法参数传入的颜色，然后调用了invalidate()方法。这段代码虽然只有三行，但是却执行了一个非常核心的功能，就是在改变了画笔颜色之后立即刷新视图，然后onDraw()方法就会调用。在onDraw()方法当中会根据当前画笔的颜色来进行绘制，这样颜色也就会动态进行改变了。

那么接下来的问题就是怎样让setColor()方法得到调用了，毫无疑问，当然是要借助ObjectAnimator类，但是在使用ObjectAnimator之前我们还要完成一个非常重要的工作，就是编写一个用于告知系统如何进行颜色过度的TypeEvaluator。创建ColorEvaluator并实现TypeEvaluator接口，代码如下所示：

**[java]** [view plain](http://blog.csdn.net/guolin_blog/article/details/43816093) [copy](http://blog.csdn.net/guolin_blog/article/details/43816093)

[在CODE上查看代码片](https://code.csdn.net/snippets/660782)

1. **public** **class** ColorEvaluator **implements** TypeEvaluator {
3. **private** **int** mCurrentRed = -1;
5. **private** **int** mCurrentGreen = -1;
7. **private** **int** mCurrentBlue = -1;
9. @Override
10. **public** Object evaluate(**float** fraction, Object startValue, Object endValue) {
11. String startColor = (String) startValue;
12. String endColor = (String) endValue;
13. **int** startRed = Integer.parseInt(startColor.substring(1, 3), 16);
14. **int** startGreen = Integer.parseInt(startColor.substring(3, 5), 16);
15. **int** startBlue = Integer.parseInt(startColor.substring(5, 7), 16);
16. **int** endRed = Integer.parseInt(endColor.substring(1, 3), 16);
17. **int** endGreen = Integer.parseInt(endColor.substring(3, 5), 16);
18. **int** endBlue = Integer.parseInt(endColor.substring(5, 7), 16);
19. // 初始化颜色的值
20. **if** (mCurrentRed == -1) {
21. mCurrentRed = startRed;
22. }
23. **if** (mCurrentGreen == -1) {
24. mCurrentGreen = startGreen;
25. }
26. **if** (mCurrentBlue == -1) {
27. mCurrentBlue = startBlue;
28. }
29. // 计算初始颜色和结束颜色之间的差值
30. **int** redDiff = Math.abs(startRed - endRed);
31. **int** greenDiff = Math.abs(startGreen - endGreen);
32. **int** blueDiff = Math.abs(startBlue - endBlue);
33. **int** colorDiff = redDiff + greenDiff + blueDiff;
34. **if** (mCurrentRed != endRed) {
35. mCurrentRed = getCurrentColor(startRed, endRed, colorDiff, 0,
36. fraction);
37. } **else** **if** (mCurrentGreen != endGreen) {
38. mCurrentGreen = getCurrentColor(startGreen, endGreen, colorDiff,
39. redDiff, fraction);
40. } **else** **if** (mCurrentBlue != endBlue) {
41. mCurrentBlue = getCurrentColor(startBlue, endBlue, colorDiff,
42. redDiff + greenDiff, fraction);
43. }
44. // 将计算出的当前颜色的值组装返回
45. String currentColor = "#" + getHexString(mCurrentRed)
46. + getHexString(mCurrentGreen) + getHexString(mCurrentBlue);
47. **return** currentColor;
48. }
50. /\*\*
51. \* 根据fraction值来计算当前的颜色。
52. \*/
53. **private** **int** getCurrentColor(**int** startColor, **int** endColor, **int** colorDiff,
54. **int** offset, **float** fraction) {
55. **int** currentColor;
56. **if** (startColor > endColor) {
57. currentColor = (**int**) (startColor - (fraction \* colorDiff - offset));
58. **if** (currentColor < endColor) {
59. currentColor = endColor;
60. }
61. } **else** {
62. currentColor = (**int**) (startColor + (fraction \* colorDiff - offset));
63. **if** (currentColor > endColor) {
64. currentColor = endColor;
65. }
66. }
67. **return** currentColor;
68. }
70. /\*\*
71. \* 将10进制颜色值转换成16进制。
72. \*/
73. **private** String getHexString(**int** value) {
74. String hexString = Integer.toHexString(value);
75. **if** (hexString.length() == 1) {
76. hexString = "0" + hexString;
77. }
78. **return** hexString;
79. }
81. }

这大概是我们整个动画操作当中最复杂的一个类了。没错，属性动画的高级用法中最有技术含量的也就是如何编写出一个合适的TypeEvaluator。好在刚才我们已经编写了一个PointEvaluator，对它的基本工作原理已经有了了解，那么这里我们主要学习一下ColorEvaluator的逻辑流程吧。

首先在evaluate()方法当中获取到颜色的初始值和结束值，并通过字符串截取的方式将颜色分为RGB三个部分，并将RGB的值转换成十进制数字，那么每个颜色的取值范围就是0-255。接下来计算一下初始颜色值到结束颜色值之间的差值，这个差值很重要，决定着颜色变化的快慢，如果初始颜色值和结束颜色值很相近，那么颜色变化就会比较缓慢，而如果颜色值相差很大，比如说从黑到白，那么就要经历255\*3这个幅度的颜色过度，变化就会非常快。

那么控制颜色变化的速度是通过getCurrentColor()这个方法来实现的，这个方法会根据当前的fraction值来计算目前应该过度到什么颜色，并且这里会根据初始和结束的颜色差值来控制变化速度，最终将计算出的颜色进行返回。

最后，由于我们计算出的颜色是十进制数字，这里还需要调用一下getHexString()方法把它们转换成十六进制字符串，再将RGB颜色拼装起来之后作为最终的结果返回。

好了，ColorEvaluator写完之后我们就把最复杂的工作完成了，剩下的就是一些简单调用的问题了，比如说我们想要实现从蓝色到红色的动画过度，历时5秒，就可以这样写：

**[java]** [view plain](http://blog.csdn.net/guolin_blog/article/details/43816093) [copy](http://blog.csdn.net/guolin_blog/article/details/43816093)

[在CODE上查看代码片](https://code.csdn.net/snippets/660782)

1. ObjectAnimator anim = ObjectAnimator.ofObject(myAnimView, "color", **new** ColorEvaluator(),
2. "#0000FF", "#FF0000");
3. anim.setDuration(5000);
4. anim.start();

用法非常简单易懂，相信不需要我再进行解释了。

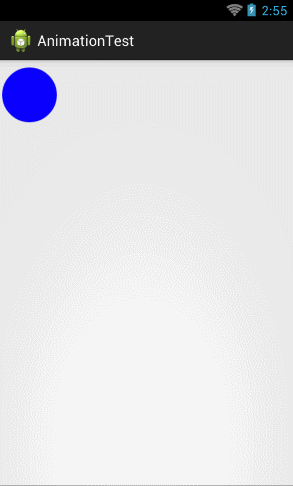
接下来我们需要将上面一段代码移到MyAnimView类当中，让它和刚才的Point移动动画可以结合到一起播放，这就要借助我们在上篇文章当中学到的组合动画的技术了。修改MyAnimView中的代码，如下所示：

**[java]** [view plain](http://blog.csdn.net/guolin_blog/article/details/43816093) [copy](http://blog.csdn.net/guolin_blog/article/details/43816093)

[在CODE上查看代码片](https://code.csdn.net/snippets/660782)

1. **public** **class** MyAnimView **extends** View {
3. ...
5. **private** **void** startAnimation() {
6. Point startPoint = **new** Point(RADIUS, RADIUS);
7. Point endPoint = **new** Point(getWidth() - RADIUS, getHeight() - RADIUS);
8. ValueAnimator anim = ValueAnimator.ofObject(**new** PointEvaluator(), startPoint, endPoint);
9. anim.addUpdateListener(**new** ValueAnimator.AnimatorUpdateListener() {
10. @Override
11. **public** **void** onAnimationUpdate(ValueAnimator animation) {
12. currentPoint = (Point) animation.getAnimatedValue();
13. invalidate();
14. }
15. });
16. ObjectAnimator anim2 = ObjectAnimator.ofObject(**this**, "color", **new** ColorEvaluator(),
17. "#0000FF", "#FF0000");
18. AnimatorSet animSet = **new** AnimatorSet();
19. animSet.play(anim).with(anim2);
20. animSet.setDuration(5000);
21. animSet.start();
22. }
24. }

可以看到，我们并没有改动太多的代码，重点只是修改了startAnimation()方法中的部分内容。这里先是将颜色过度的代码逻辑移动到了startAnimation()方法当中，注意由于这段代码本身就是在MyAnimView当中执行的，因此ObjectAnimator.ofObject()的第一个参数直接传this就可以了。接着我们又创建了一个AnimatorSet，并把两个动画设置成同时播放，动画时长为五秒，最后启动动画。现在重新运行一下代码，效果如下图所示：



OK，位置动画和颜色动画非常融洽的结合到一起了，看上去效果还是相当不错的，这样我们就把ObjectAnimator的高级用法也掌握了。

好的，通过本篇文章的学习，我们对属性动画已经有了颇为深刻的认识，那么本篇文章的内容到此为止，下篇文章当中将会介绍更多关于属性动画的其它技巧，感兴趣的朋友请继续阅读 [**Android属性动画完全解析(下)，Interpolator和ViewPropertyAnimator的用法**](http://blog.csdn.net/guolin_blog/article/details/44171115) 。