**ObjectAnimator Вычислители**

В Android версии API 11 (и добавлены новые в API 18) появились вычислители (evaluator), которые применимы к классу **ObjectAnimator** и позволяют задать нужную анимацию цвета, местоположения, размера.

[ArgbEvaluator](http://developer.alexanderklimov.ru/android/animation/objectanimator2.php#argb)  
[RectEvaluator](http://developer.alexanderklimov.ru/android/animation/objectanimator2.php#rect)

В [первой части](http://developer.alexanderklimov.ru/android/animation/objectanimator.php) для анимации использовался метод **ofFloat()**. Существует родственный ему метод **ofInt()**, работающий с целочисленными значениями. Но на этом способы анимации не заканчиваются. Вы можете оперировать не только числовыми значениями, но и объектами цвета **ARGB** или прямоугольника **Rect** с помощью классов **ArgbEvaluator** и **RectEvaluator** соответственно. А также вы можете создать свой собственный класс вычислителя

Возвращаясь к старым примерам, нужно заметить, что под капотом использовались классы **FloatEvaluator** и **IntEvaluator**, которые брали значения от интерполяторов и использовали их в своих вычислениях для анимации.

При работе с новыми вычислителями, о которых пойдёт речь, используется метод **ofObject()**.

**ArgbEvaluator**

Начнём с вычислителя **ArgbEvaluator**. Изменим цвет кнопки при щелчке.

public void onClick(View v) {

ObjectAnimator animator = ObjectAnimator.ofObject(button,

"backgroundColor", new ArgbEvaluator(), Color.RED, Color.BLUE);

animator.setDuration(2000);

animator.start();

}

Мы указали тип анимации **backgroundColor** и вычислитель **ArgbEvaluator** с начальным и конечным значениями цветов. При нажатии кнопка будет переливаться из красного в синий.

Метод может содержать различное количество параметров в конце после указания класса **ArgbEvaluator**. Например, мы можем указать не два, а три цвета. И кнопка будет менять свой цвет, проходя через указанные значения.

public void onClick(View v) {

ObjectAnimator animator = ObjectAnimator.ofObject(button,

"backgroundColor", new ArgbEvaluator(), Color.RED, Color.BLUE, Color.YELLOW);

animator.setDuration(2000);

animator.start();

}

Теперь кнопка переливается от красного в жёлтый, на время становясь синей.

**RectEvaluator**

Теперь рассмотрим пример с **RectEvaluator**. Напомню, что он доступен с API 18. Для анимации с помощью этого вычислителя используется свойство **clipBounds** выбранного компонента.

Применим анимацию к **ImageView**.

Rect rect = new Rect();

mImageView.getLocalVisibleRect(rect);

Rect from = new Rect(rect);

Rect to = new Rect(rect);

from.right = from.left + rect.width() / 4;

from.bottom = from.top + rect.height() / 2;

to.left = to.right - rect.width() / 2;

to.top = to.bottom - rect.height() / 4;

ObjectAnimator rectAnimator = ObjectAnimator.ofObject(mImageView,

"clipBounds", new RectEvaluator(), from, to);

rectAnimator.setDuration(2000);

rectAnimator.start();

Для данного случая нам нужно выбрать не цвета, а прямоугольники. С помощью метода **getLocalVisibleRect()** мы получаем видимую часть прямоугольника у **ImageView**. Затем создаются две копии прямоугольника и меняются их размеры. Первый прямоугольник послужит отправной точкой для анимации и будет располагаться в верхнем левом углу в половину высоты и четверть ширины. Конечный прямоугольник находится в нижнем правом углу с другими пропорциями.

Далее выбирается тип анимации **clipBounds** и вычислитель **RectEvaluator**.

**Интерфейс TypeEvaluator<T>**

Если возможностей стандартных вычислителей вам не достаточно, то можете создать свой собственный вычислитель на основе интерфейса **TypeEvaluator<T>** с реализацией метода **evaluate()**.

public T evaluate(float fraction, T startValue, T endValue);

Вам нужно определить начальное и конечное значение и возвратить значение для интерполятора. Параметр **fraction** определяет позицию для интерполятора. Если он равен 0.0, то должен соответствовать начальному значению, если равен 1.0, то - конечному. Промежуточные значения должные возвращать объект, лежащий между двумя крайними значениями.

Создадим вычислитель **MatrixEvaluator**, который будет использовать матричные преобразования для создания эффекта **Skew**.

package ru.alexanderklimov.test;

import android.animation.TypeEvaluator;

import android.graphics.Matrix;

public class MatrixEvaluator implements TypeEvaluator<Matrix> {

public Matrix evaluate(float fraction, Matrix startValue, Matrix endValue) {

float[] startEntries = new float[9];

float[] endEntries = new float[9];

float[] currentEntries = new float[9];

startValue.getValues(startEntries);

endValue.getValues(endEntries);

for (int i = 0; i < 9; i++)

currentEntries[i] = (1 - fraction) \* startEntries[i] + fraction

\* endEntries[i];

Matrix matrix = new Matrix();

matrix.setValues(currentEntries);

return matrix;

}

}

Применим вычислитель к **ImageView**:

mImageView2.setScaleType(ScaleType.MATRIX);

float scale = (float) mImageView2.getHeight()

/ (float) mImageView2.getDrawable().getIntrinsicHeight();

Matrix fromMatrix = new Matrix();

fromMatrix.setScale(scale, scale);

fromMatrix.postSkew(-0.5f, 0.0f);

Matrix toMatrix = new Matrix(mImageView2.getMatrix());

toMatrix.setScale(scale, scale);

toMatrix.postSkew(0.5f, 0.0f);

ObjectAnimator matrixAnimator = ObjectAnimator.ofObject(

mImageView2, "imageMatrix", new MatrixEvaluator(),

fromMatrix, toMatrix);

matrixAnimator.setDuration(500);

matrixAnimator.setRepeatCount(4);

matrixAnimator.setRepeatMode(ObjectAnimator.REVERSE);

matrixAnimator.start();

Сначала мы установим тип **ScaleType.MATRIX**, также можно установить этот тип через XML-атрибут **android:scaleType**. Далее устанавливаются матричные преобразования и вызываются методы **setScale()** и **postSkew()**.

Свойство **imageMatrix** связано с методом **setImageMatrix()**. Вызываем метод **ofObject()** с подготовленными параметрами, устанавливаем продолжительность анимации и число повторов, а также обратную анимацию. Новый эффект готов.

**Используем другую цветовую модель**

Если вас не устраивает цветовая модель RGB в вычислителе **ArgbEvaluator**, то можете создать свой вычислитель на основе модели HSV.

package ru.alexanderklimov.test;

import android.animation.TypeEvaluator;

import android.graphics.Color;

public class HsvEvaluator implements TypeEvaluator<Integer> {

public Integer evaluate(float fraction, Integer startValue, Integer endValue) {

float[] startHsv = new float[3];

float[] endHsv = new float[3];

float[] currentHsv = new float[3];

Color.colorToHSV(startValue, startHsv);

Color.colorToHSV(endValue, endHsv);

for (int i = 0; i < 3; i++)

currentHsv[i] = (1 - fraction) \* startHsv[i] + fraction \* endHsv[i];

while (currentHsv[0] >= 360.0f)

currentHsv[0] -= 360.0f;

while (currentHsv[0] < 0.0f)

currentHsv[0] += 360.0f;

return Color.HSVToColor(currentHsv);

}

}

Применим эффект ко второй кнопке.

Button button2 = (Button) findViewById(R.id.button2);

ObjectAnimator hsvAnimator = ObjectAnimator.ofObject(button2,

"backgroundColor", new HsvEvaluator(), Color.RED,

Color.BLUE);

hsvAnimator.setDuration(2000);

hsvAnimator.start();

**Использованные материалы**

[Android Property Animations: Evaluators](http://cogitolearning.co.uk/?p=1451)