29 апреля в 19:14

Разработка → Animator — что это? Зачем он нужен? Почему его стоит использовать вместо Animation?

Разработка под Android*, JAVA*, Блог компании Лайв Тайпинг

Привет! Меня зовут Данил Перевалов, я работаю Android-разработчиком в компании «Лайв Тайпинг», и сегодня мне выпала честь открыть наш блог на Хабре. Дебютный материал посвящён типу классов Animator. Статья будет полезна и тем, кто только столкнулся с созданием анимаций, и тем, кому недостаточно уже имеющихся знаний по предмету. Тех же, кто давно знаком с темой, мы настойчиво просим делиться опытом в комментариях.

Что такое — Animator?

Немного истории. С момента запуска платформы Android существовал фреймворк View Animation. Предназначался он, как следует из названия, для анимаций. Но производительность устройств в конце нулевых была настолько низкой, что о красивых анимациях никто особо не думал, поэтому фреймворк не был удобным и гибким. Он имел только четыре типа анимации (TranslateAnimation, AlphaAnimation, ScaleAnimation, RotateAnimation), класс, позволяющий их комбинировать (AnimationSet), а также способность работать только с классами, унаследованными от View.

В Android 3.0 появился куда более гибкий фреймворк Property Animation. Он умеет изменять любое доступное свойство, а также может работать с любыми классами. Его основным инструментом является Animator.

Animator — это тип классов, предназначенных для изменения значений выбранного объекта относительно времени. Грубо говоря, это инструмент для управления потоком заданной длительности, который изменяет определённое свойство от начального значения к конечному. Таким плавно меняющимся свойством в анимации может быть, например, прозрачность.

Идеологическое отличие классов Animator от View Animation в том, что View Animation изменяет «представление» объекта, не изменяя сам объект (исключением является использование setFillAfter(true), но с этим флагом объект изменяется в конце анимации). Animator же призван изменять свойства самого объекта.

Классы, унаследованные от Animator

ValueAnimator (наследуется от Animator)

В самом простом варианте мы задаём этому классу тип изменяемого значения, начальное значение и конечное значение, и запускаем. В ответ нам будут приходить события на начало, конец, повторение и отмену анимации и ещё на два события, которые задаются отдельно для паузы и изменения значения. Событие изменения, пожалуй, самое важное: в него будет приходить изменённое значение, с помощью которого мы и будем менять свойства объектов.

Посмотрите на изменение alpha с его помощью:

```
ValueAnimator animator = ValueAnimator.ofFloat(0, 1);
animator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
   @Override
   public void onAnimationUpdate(ValueAnimator animation) {
       view.setAlpha((Float) animation.getAnimatedValue());
   }
});
animator.start();
```

ObjectAnimator, наследуется от ValueAnimator

Это класс, призванный упростить работу с ValueAnimator. С ним вам не нужно вручную изменять какое-либо значение по событию изменения — вы просто даёте Animator'у объект и указываете поле, которое вы хотите изменить, например scaleX. С помощью Java Refliction ищется setter для этого поля (в данном случае — setScaleX.

Далее Animator самостоятельно будет менять значение этого поля.
С помощью ObjectAnimator изменение alpha будет выглядеть так:

```
ObjectAnimator.ofFloat(view, View.ALPHA, 0, 1).start();
```

У класса View есть несколько свойств специально предназначенных для анимирования с помощью Animator:

- прозрачность (View.ALPHA)
- масштаб (View.SCALE_X, View.SCALE_Y)
- вращение (View.ROTATION, View.ROTATION_X, View.ROTATION_Y)
- положение (View.X, View.Y, View.Z)
- положение отображаемой части (View.TRANSLATION_X, View.TRANSLATION_Y, View.TRANSLATION_Z)

AnimatorSet (наследуется от Animator)

Это класс, позволяющий комбинировать анимации различными способами: запускать одновременно или последовательно, добавлять задержки и т.д.

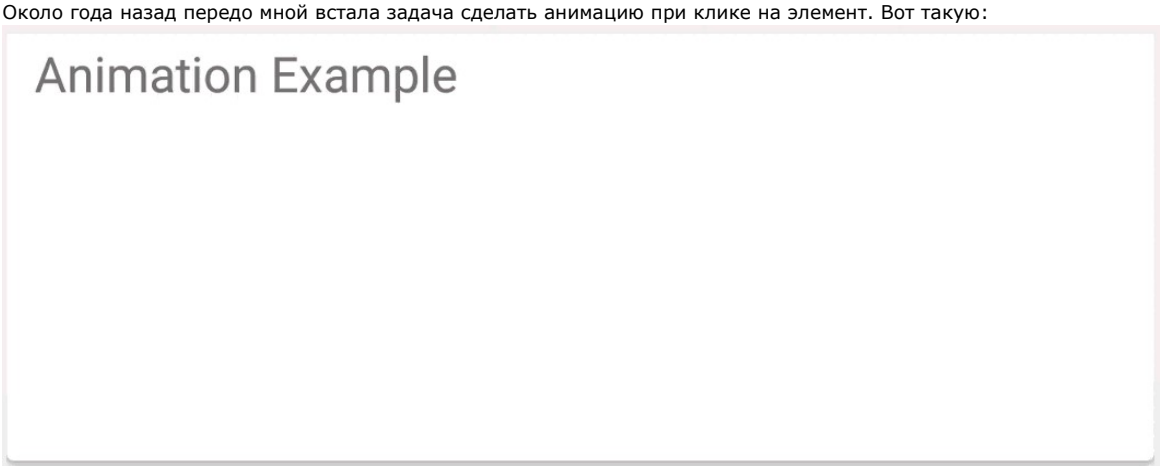
ViewPropertyAnimator

Это отдельный класс. Он не наследуется от Animator, но обладает той же логикой, что и ObjectAnimator для View, и предназначен для лёгкого анимирования какой-либо View без лишних заморочек.
Вот так с его помощью можно изменить alpha:

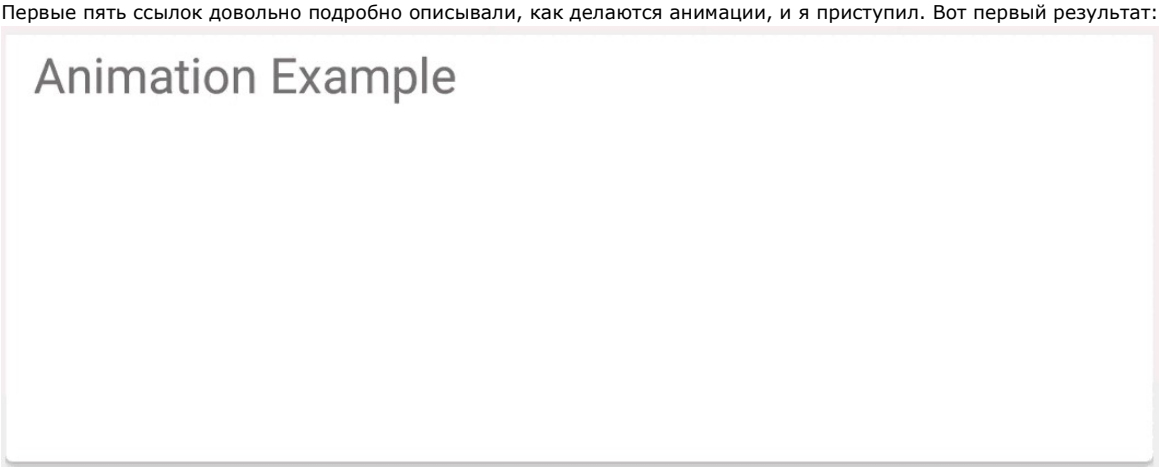
```
view.animate().alphaBy(0).alpha(1).start();
```

Как мы начали использовать Animator

Около года назад передо мной встала задача сделать анимацию при клике на элемент. Вот такую:



Не то чтобы я не делал анимаций прежде, но на аутсорсе они редко нужны. Поэтому я загуглил Animation Android. Первые пять ссылок довольно подробно описывали, как делаются анимации, и я приступил. Вот первый результат:



Код Animation

```
public static void likeAnimation(@DrawableRes int icon,
```

```
                                    final ImageView imageView) {
        imageView.setImageResource(icon);
        imageView.setVisibility(View.VISIBLE);
        AlphaAnimation showAlphaAnimation = new AlphaAnimation(0.0f, 1.0f);
        showAlphaAnimation.setDuration(SHOW_DURATION);
        ScaleAnimation showScaleAnimation = new ScaleAnimation(0.2f, 1.4f, 0.2f, 1.4f,
                android.view.animation.Animation.RELATIVE_TO_SELF, 0.5f,
                android.view.animation.Animation.RELATIVE_TO_SELF, 0.5f);
        showScaleAnimation.setDuration(SHOW_DURATION);
        AnimationSet showAnimationSet = new AnimationSet(false);
        showAnimationSet.addAnimation(showAlphaAnimation);
        showAnimationSet.addAnimation(showScaleAnimation);
        showAnimationSet.setAnimationListener(new OnEndAnimationListener() {
            @Override
            public void onAnimationEnd(android.view.animation.Animation animation) {
                ScaleAnimation toNormalScaleAnimation = new ScaleAnimation(1.4f, 1.0f, 1.4f, 1.0f,
                        android.view.animation.Animation.RELATIVE_TO_SELF, 0.5f,
                        android.view.animation.Animation.RELATIVE_TO_SELF, 0.5f);
                toNormalScaleAnimation.setDuration(TO_NORMAL_DURATION);
                toNormalScaleAnimation.setAnimationListener(new OnEndAnimationListener() {
                    @Override
                    public void onAnimationEnd(android.view.animation.Animation animation) {
                        AlphaAnimation hideAlphaAnimation = new AlphaAnimation(1.0f, 0.0f);
                        hideAlphaAnimation.setDuration(HIDE_DURATION);
                        ScaleAnimation hideScaleAnimation = new ScaleAnimation(1.0f, 0.2f, 1.0f, 0.2f,
                                android.view.animation.Animation.RELATIVE_TO_SELF, 0.5f,
                                android.view.animation.Animation.RELATIVE_TO_SELF, 0.5f);
                        hideScaleAnimation.setDuration(HIDE_DURATION);
                        AnimationSet hideAnimationSet = new AnimationSet(false);
                        hideAnimationSet.setStartOffset(HIDE_DELAY);
                        hideAnimationSet.addAnimation(hideAlphaAnimation);
                        hideAnimationSet.addAnimation(hideScaleAnimation);
                        hideAnimationSet.setAnimationListener(new OnEndAnimationListener() {
                            @Override
                            public void onAnimationEnd(android.view.animation.Animation animation) {
                                imageView.setVisibility(View.GONE);
                            }
                        });
                        imageView.startAnimation(hideAnimationSet);
                    }
                });
                imageView.startAnimation(toNormalScaleAnimation);
            }
        });
        imageView.startAnimation(showAnimationSet);
    }
```

Ссылка на код

Код получился малопонятным, что подтолкнуло меня к поискам иного подхода в составлении последовательности анимаций. Решение было найдено на StackOveflow. Идея такая: помещать в последовательности анимаций каждую последующую анимацию в AnimationSet со сдвигом, равным сумме длительностей предыдущих анимаций. Получилось гораздо лучше, чем было:

Animation Set Offset Example

AnimationSet

```
public static void likeAnimation(@DrawableRes int icon,
                                 final ImageView imageView) {
        imageView.setImageResource(icon);
        imageView.setVisibility(View.VISIBLE);
        AnimationSet animationSet = new AnimationSet(false);
        animationSet.addAnimation(showAlphaAnimation());
        animationSet.addAnimation(showScaleAnimation());
        animationSet.addAnimation(toNormalScaleAnimation());
        animationSet.addAnimation(hideAlphaAnimation());
        animationSet.addAnimation(hideScaleAnimation());
        animationSet.setAnimationListener(new OnEndAnimationListener() {
            @Override
            public void onAnimationEnd(Animation animation) {
                imageView.setVisibility(View.GONE);
            }
        });
        imageView.startAnimation(animationSet);
    }

    private static Animation showAlphaAnimation() {
        AlphaAnimation showAlphaAnimation = new AlphaAnimation(0.0f, 1.0f);
        showAlphaAnimation.setDuration(SHOW_DURATION);
        return showAlphaAnimation;
    }

    private static Animation showScaleAnimation() {
        ScaleAnimation showScaleAnimation = new ScaleAnimation(
                0.2f, 1.4f, 0.2f, 1.4f,
                Animation.RELATIVE_TO_SELF, 0.5f,
                Animation.RELATIVE_TO_SELF, 0.5f);
        showScaleAnimation.setDuration(SHOW_DURATION);
        return showScaleAnimation;
    }

    private static Animation toNormalScaleAnimation() {
        ScaleAnimation toNormalScaleAnimation = new ScaleAnimation(
                1.4f, 1.0f, 1.4f, 1.0f,
                Animation.RELATIVE_TO_SELF, 0.5f,
                Animation.RELATIVE_TO_SELF, 0.5f);
        toNormalScaleAnimation.setDuration(TO_NORMAL_DURATION);
        toNormalScaleAnimation.setStartOffset(SHOW_DURATION);
        return toNormalScaleAnimation;
    }

    private static Animation hideAlphaAnimation() {
        AlphaAnimation hideAlphaAnimation = new AlphaAnimation(1.0f, 0.0f);
        hideAlphaAnimation.setDuration(HIDE_DURATION);
        hideAlphaAnimation.setStartOffset(SHOW_DURATION + TO_NORMAL_DURATION + HIDE_DELAY);
        return hideAlphaAnimation;
```

```
    }

    private static Animation hideScaleAnimation() {
        ScaleAnimation hideScaleAnimation = new ScaleAnimation(
                1.0f, 0.2f, 1.0f, 0.2f,
                Animation.RELATIVE_TO_SELF, 0.5f,
                Animation.RELATIVE_TO_SELF, 0.5f);
        hideScaleAnimation.setDuration(HIDE_DURATION);
        hideScaleAnimation.setStartOffset(SHOW_DURATION + TO_NORMAL_DURATION + HIDE_DELAY);
        return hideScaleAnimation;
    }
```

Ссылка на код

Код стал понятнее и читабельнее, но есть одно «но»: следить за сдвигом у каждой анимации довольно неудобно даже в такой простой последовательности. Если добавить ещё несколько шагов, то это станет почти невыполнимой задачей. Также важным минусом такого подхода стало странное поведение анимации: размер анимированного объекта, по непонятным для меня причинам, был больше, чем при обычной последовательности анимаций. Попытки разобраться ни к чему не привели, а вникать глубже я уже не стал — подход мне всё равно не нравился. Но я решил развить эту идею и разбить каждый шаг на отдельный AnimatorSet. Вот что вышло:

AnimatorSet в AnimatorSet

```
public static void likeAnimation(@DrawableRes int icon,
                                 final ImageView imageView) {
        imageView.setImageResource(icon);
        imageView.setVisibility(View.VISIBLE);
        AnimationSet animationSet = new AnimationSet(false);
        animationSet.addAnimation(showAnimationSet());
        animationSet.addAnimation(toNormalAnimationSet());
        animationSet.addAnimation(hideAnimationSet());
        animationSet.setAnimationListener(new OnEndAnimationListener() {
            @Override
            public void onAnimationEnd(Animation animation) {
                imageView.setVisibility(View.GONE);
            }
        });
        imageView.startAnimation(animationSet);
    }

    private static AnimationSet showAnimationSet() {
        AlphaAnimation showAlphaAnimation = new AlphaAnimation(0.0f, 1.0f);
        ScaleAnimation showScaleAnimation = new ScaleAnimation(
                0.2f, 1.4f, 0.2f, 1.4f,
                Animation.RELATIVE_TO_SELF, 0.5f,
                Animation.RELATIVE_TO_SELF, 0.5f);
        AnimationSet set = new AnimationSet(false);
        set.addAnimation(showAlphaAnimation);
        set.addAnimation(showScaleAnimation);
        set.setDuration(SHOW_DURATION);
```

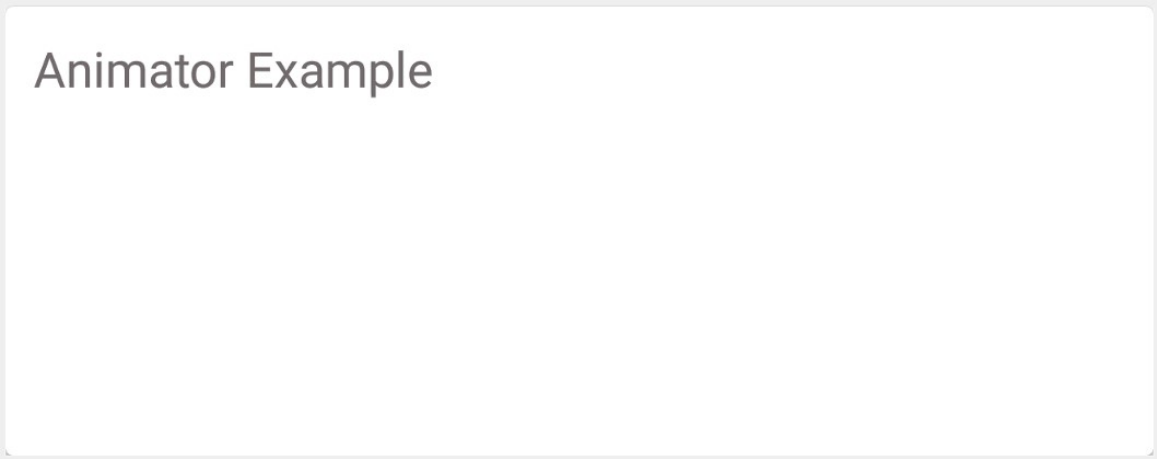
```
        return set;
    }

    private static AnimationSet toNormalAnimationSet() {
        ScaleAnimation toNormalScaleAnimation = new ScaleAnimation(
                1.4f, 1.0f, 1.4f, 1.0f,
                Animation.RELATIVE_TO_SELF, 0.5f,
                Animation.RELATIVE_TO_SELF, 0.5f);
        AnimationSet set = new AnimationSet(false);
        set.addAnimation(toNormalScaleAnimation);
        set.setDuration(TO_NORMAL_DURATION);
        set.setStartOffset(SHOW_DURATION);
        return set;
    }

    private static AnimationSet hideAnimationSet() {
        AlphaAnimation hideAlphaAnimation = new AlphaAnimation(1.0f, 0.0f);
        ScaleAnimation hideScaleAnimation = new ScaleAnimation(
                1.0f, 0.2f, 1.0f, 0.2f,
                Animation.RELATIVE_TO_SELF, 0.5f,
                Animation.RELATIVE_TO_SELF, 0.5f);
        AnimationSet set = new AnimationSet(false);
        set.setDuration(HIDE_DURATION);
        set.addAnimation(hideAlphaAnimation);
        set.addAnimation(hideScaleAnimation);
        set.setStartOffset(SHOW_DURATION + TO_NORMAL_DURATION + HIDE_DELAY);
        return set;
    }
```

Ссылка на код

Некорректная работа анимации, плохой подход, всё плохо. Вновь я обратился к Google, и наткнулся на то, что Animation уже является Legacy code, то есть устарел и не поддерживается, хотя и используется.
Я понял, что нужно делать анимации совершенно иначе. И вот на просторах Android Developers я наткнулся на Animator. Попытка сделать анимацию с его помощью выглядела так:



Animator

```
public static void likeAnimation(@DrawableRes int icon,
                                 final ImageView view) {
        if (view != null && !isAnimate) {
            AnimatorSet set = new AnimatorSet();
            set.playSequentially(
                    showAnimatorSet(view),
                    toNormalAnimatorSet(view),
                    hideAnimatorSet(view));
            set.addListener(getLikeEndListener(view, icon));
            set.start();
        }
        view.animate().alphaBy(0).alpha(1).start();
```

```
    }

    private static AnimatorListenerAdapter getLikeEndListener(final ImageView view, final int icon) {
        return new AnimatorListenerAdapter() {
            @Override
            public void onAnimationStart(Animator animation) {
                super.onAnimationStart(animation);
                isAnimate = true;
                view.setVisibility(View.VISIBLE);
                view.setImageResource(icon);
                view.setLayerType(View.LAYER_TYPE_HARDWARE, null);
            }

            @Override
            public void onAnimationEnd(Animator animation) {
                super.onAnimationEnd(animation);
                isAnimate = false;
                view.setVisibility(View.GONE);
                view.setImageDrawable(null);
                view.setLayerType(View.LAYER_TYPE_NONE, null);
            }
        };
    }

    private static AnimatorSet showAnimatorSet(View view) {
        AnimatorSet set = new AnimatorSet();
        set.setDuration(SHOW_DURATION).playTogether(
                ObjectAnimator.ofFloat(view, View.ALPHA, 0f, 1f),
                ObjectAnimator.ofFloat(view, View.SCALE_X, 0.2f, 1.4f),
                ObjectAnimator.ofFloat(view, View.SCALE_Y, 0.2f, 1.4f)
        );
        return set;
    }

    private static AnimatorSet toNormalAnimatorSet(View view) {
        AnimatorSet set = new AnimatorSet();
        set.setDuration(TO_NORMAL_DURATION).playTogether(
                ObjectAnimator.ofFloat(view, View.SCALE_X, 1.4f, 1f),
                ObjectAnimator.ofFloat(view, View.SCALE_Y, 1.4f, 1f)
        );
        return set;
    }

    private static AnimatorSet hideAnimatorSet(View view) {
        AnimatorSet set = new AnimatorSet();
        set.setDuration(HIDE_DURATION).playTogether(
                ObjectAnimator.ofFloat(view, View.ALPHA, 1f, 0f),
                ObjectAnimator.ofFloat(view, View.SCALE_X, 1f, 0.2f),
                ObjectAnimator.ofFloat(view, View.SCALE_Y, 1f, 0.2f)
        );
        set.setStartDelay(HIDE_DELAY);
        return set;
    }
```

Ссылка на код

Анимация работала безупречно, а значит, поиски можно считать оконченными. Единственное, что при работе с Animator нужно помнить, не запущен ли уже какой-то Animator для конкретной view, потому что в противном случае старый продолжит выполнятся, как ни в чем не бывало.

Глубже в Animator

Я начал поиски того, что ещё интересного можно сделать с помощью Animator. Полёт мысли привёл меня к следующему:

При нажатии на кнопку одновременно выполняется четыре Animator'а:

▾ Одновременный запуск

```
AnimatorSet showHideSet = new AnimatorSet();
showHideSet.playTogether(
          ScrollAnimatorUtils.translationYAnimator(translationY, footerButtons),
          ScrollAnimatorUtils.translationYAnimator(translationY, footerText),
          ScrollAnimatorUtils.scrollAnimator(startScroll, endScroll, scrollView),
          ScrollAnimatorUtils.alphaAnimator(1, 0, recyclerView)
  );
  showHideSet.start();
```

1) двигает вниз footer списка;
2) двигает вниз кнопки;

▾ translationYAnimator

```
public static Animator translationYAnimator(final float start, int end, final View view, int duration) {
       ObjectAnimator animator = ObjectAnimator.ofFloat(view, View.TRANSLATION_Y, end);
       animator.setDuration(duration);
       animator.addListener(new AnimatorListenerAdapter() {
           @Override
           public void onAnimationEnd(Animator animation) {
               super.onAnimationEnd(animation);
               view.setTranslationY(start);
           }
       });
       return animator;
   }
```

3) скроллит ScrollView до самого низа;

▾ scrollAnimator

```
public static Animator scrollAnimator(int start, int end, final View view, int duration) {
        ValueAnimator scrollAnimator = ValueAnimator.ofInt(start, end);
        scrollAnimator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
            @Override
            public void onAnimationUpdate(ValueAnimator valueAnimator) {
                view.scrollTo(0, (int) valueAnimator.getAnimatedValue());
            }
        });
        scrollAnimator.setDuration(duration);
        scrollAnimator.addListener(getLayerTypeListener(view));
        return scrollAnimator;
    }
```

4) накладывает alpha эффект на recyclerView.

alphaAnimator

```
public static Animator alphaAnimator(int start, int end, View view, int duration) {
        ValueAnimator alphaAnimator = ObjectAnimator.ofFloat(view, View.ALPHA, start, end);
        alphaAnimator.setDuration(duration);
        alphaAnimator.addListener(getLayerTypeListener(view));
        return alphaAnimator;
    }
```

Ссылка на полный код

На этом краткий обзор Animator закончен. Если вы в курсе каких-то интересных приёмов, связанных с Animator — дополняйте статью своими комментариями.
Ссылка на проект в Github: github.com/princeparadoxes/AnimationVsAnimator

Android, animation, animator

Комментарии (5)

arturdumchev 29 апреля 2016 в 21:14 0

Неплохая библиотека на основе аниматоров.

Если кто будет использовать, обратите внимание, что вместо обычных аниматоров нужно импортировать

```
import com.nineoldandroids.animation.Animator
import com.nineoldandroids.animation.AnimatorSet
import com.nineoldandroids.animation.ObjectAnimator
```

ответить

7voprosov 30 апреля 2016 в 02:17 +3

nineoldandroid — уже 4 года как deprecated. Да и сама ваша библиотека почти как год не обновлялась. Такие библиотеки все же лучше не тащить в проект (как по мне). В конечном итоге можно посмотреть «как сделано» и сделать так же/вытащить нужный кусок (да-да, копипаста ужас ужас).

ответить

thelongrunsmoke 30 апреля 2016 в 06:06 0

В старых проектах, эта библиотека встречается ещё очень часто. Однако, тянуть её в новые разработки, конечно, не стоит. Почти весь функционал покрыт библиотеками обратной совместимости.

ответить

arturdumchev 9 мая 2016 в 18:24 0

Вот попалась библиотека. Все те же эффекты, нужно только interpolator поставить.

ответить

Beanut 30 апреля 2016 в 11:14 *(комментарий был изменён)* +2

Как вариант, можно было свой кастомный BounceInterpolator сделать. Тут можно поэкспериментировать: http://inloop.github.io/interpolator/

ответить

Вы не можете комментировать эту публикацию

Можно комментировать публикации, которые не старше 3-х дней, а также те, которые вы уже комментировали ранее.

Самое читаемое

Сейчас Неделя Месяц

+8 Шкура неубитого единорога: юридические разбороки со стартапом-миллиардером Cruise
5k 6 2

+9 Как отслеживать новости в мире C++
1,9k 42 3

+4 Google превращает свои дата центры в произведения искусства
2,2k 3 0

+46 Совсем не нейронные сети
18,3k 198 74

+15 Запуск NodeJS-приложения на Android
5,4k 39 11

Интересные публикации

- Графический интерфейс к демону tacacs — TacacsGUI 0
- Как отслеживать новости в мире C++ 3
- Как мы рисовали road shields на карте 0
- 26-летний мужчина получил протез руки с зарядкой для телефона, фонариком и дроном 23
- Планировщик путешествий своими руками за пару часов 1