属性动画概览

属性动画系统是一个强健的框架，用于为几乎任何内容添加动画效果。您可以定义一个随时间更改任何对象属性的动画，无论其是否绘制到屏幕上。属性动画会在指定时长内更改属性（对象中的字段）的值。要添加动画效果，请指定要添加动画效果的对象属性，例如对象在屏幕上的位置、动画效果持续多长时间以及要在哪些值之间添加动画效果。

借助属性动画系统，您可以定义动画的以下特性：

- 时长：您可以指定动画的时长。默认时长为 300 毫秒。
- 时间插值：您可以指定如何根据动画的当前已播放时长来计算属性的值。
- 重复计数和行为：您可以指定是否在某个时长结束后重复播放动画以及重复播放动画多少次。您还可以指定是否要反向播放动画。如果将其设置为反向播放，则会先播放动画，然后反向播放动画，直到达到重复次数。
- Animator 集：您可以将动画分成多个逻辑集，它们可以一起播放、按顺序播放或者在指定的延迟时间后播放。
- 帧刷新延迟：您可以指定动画帧的刷新频率。默认设置为每 10 毫秒刷新一次，但应用刷新帧的速度最终取决于整个系统的繁忙程度以及系统为底层计时器提供服务的速度。

要查看属性动画的完整示例，请参阅 GitHub 上 CustomTransition (https://github.com/android/animation-samples/tree/master/CustomTransition) 示例中的 `ChangeColor` 类。

属性动画的工作原理

首先，让我们通过一个简单的示例来了解动画的工作原理。图 1 描绘了一个假设的对象，该对象的 x 属性（表示其在屏幕上的水平位置）添加了动画效果。动画时长设置为 40 毫秒，要移动的距离为 40 像素。该对象每隔 10 毫秒（这是默认的帧刷新频率）会水平移动 10 像素。在 40 毫秒时，动画停止，同时对象在水平位置 40 处停止。这是使用线性插值（表示对象以恒定速度移动）的动画示例。

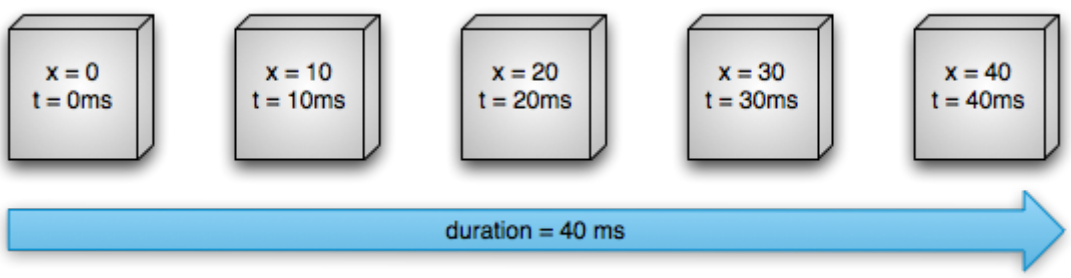

图 1. 线性动画示例

您也可以指定动画使用非线性插值。图 2 展示了一个假设的对象，它在动画开始时加速，在动画结束前减速。该对象仍在 40 毫秒内移动了 40 像素，但这种移动是非线性的。开始时，此动画加速移动到中间点，然后从中间点减速移动，直至动画结束。如图 2 所示，动画在开头和结尾移动的距离小于在中间移动的距离。

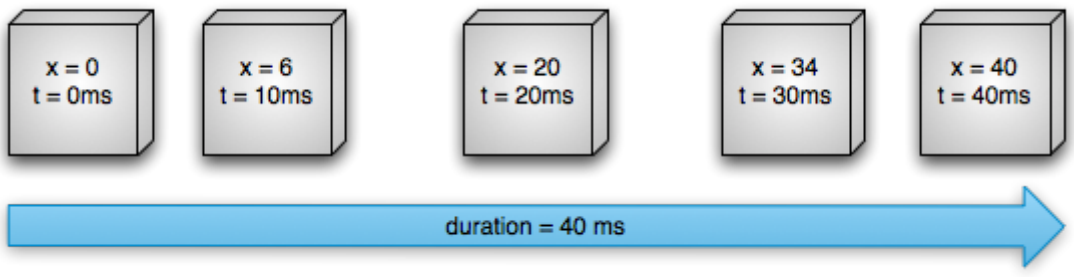

图 2. 非线性动画示例

我们来详细了解一下属性动画系统的重要组成部分将如何计算如上所示的动画。图 3 描绘了主类之间是如何相互协作的。

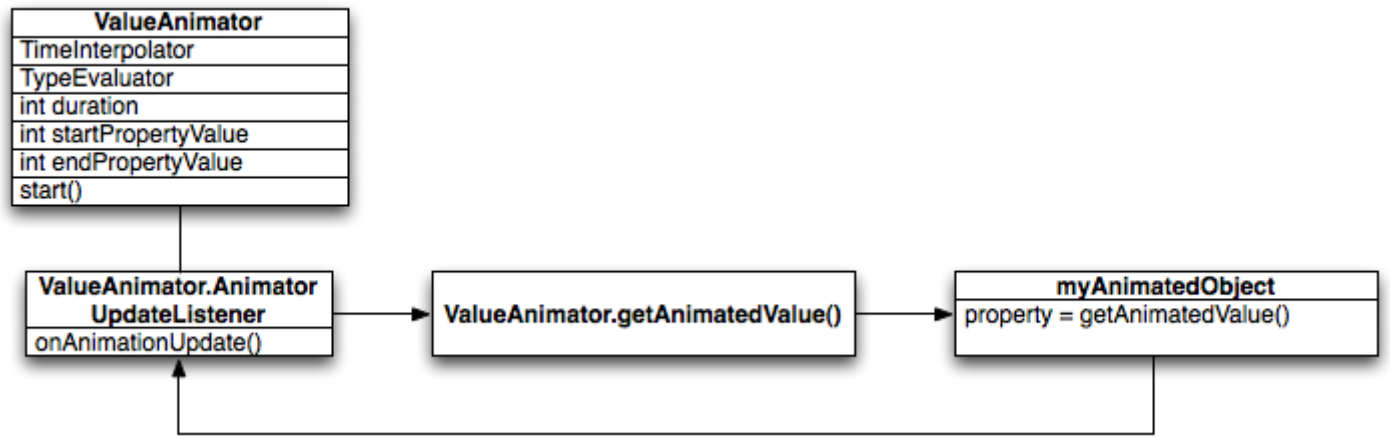

图 3. 如何计算动画

`ValueAnimator` (/reference/android/animation/ValueAnimator) 对象跟踪动画的时间，例如动画的已运行时长以及正在添加动画效果的属性的当前值。

`ValueAnimator` (/reference/android/animation/ValueAnimator) 包含 `TimeInterpolator` (/reference/android/animation/TimeInterpolator) 和 `TypeEvaluator` (/reference/android/animation/TypeEvaluator)；前者用于定义动画插值，后者用于定义如何计算正在添加动画效果的属性的值。例如，在图 2 中，所用的 `TimeInterpolator` (/reference/android/animation/TimeInterpolator) 为 `AccelerateDecelerateInterpolator` (/reference/android/view/animation/AccelerateDecelerateInterpolator)，所用的 `TypeEvaluator` (/reference/android/animation/TypeEvaluator) 为 `IntEvaluator` (/reference/android/animation/IntEvaluator)。

要开始动画，请创建一个 `ValueAnimator` (/reference/android/animation/ValueAnimator)，并为您想要添加动画效果的属性赋予起始值和结束值，以及动画时长。当您调用 `start()` (/reference/android/animation/ValueAnimator#start()) 时，动画即会开始播放。在整个动画播放期间，`ValueAnimator` (/reference/android/animation/ValueAnimator) 将基于动画时长和已播放时长计算已完成动画分数（在 0 和 1 之间）。已完成动画分数表示动画已完成时间的百分比，0 表示 0%，1 表示 100%。以图 1 为例，在 t = 10ms 处，已完成动画分数将为 0.25，因为总时长 t = 40ms。

在 `ValueAnimator` (/reference/android/animation/ValueAnimator) 计算完已完成动画分数后，它会调用当前设置的 `TimeInterpolator` (/reference/android/animation/TimeInterpolator) 来计算插值分数。插值分数会将已完成动画分数映射为一个新分数，该分数会考虑已设置的时间插值。例如，在图 2 中，由于动画缓慢加速，t = 10ms 时的插值分数（约 0.15）小于已完成动画分数 (0.25)。在图 1 中，插值分数始终等于已完成动画分数。

计算插值分数后，`ValueAnimator` (/reference/android/animation/ValueAnimator) 会调用相应的 `TypeEvaluator` (/reference/android/animation/TypeEvaluator)，以根据动画的插值分数、起始值和结束值来计算要添加动画效果的属性的值。例如，在图 2 中，t = 10ms 时的插值分数为 0.15，因此，此时属性的值为 0.15 × (40 - 0)，即 6。

属性动画与视图动画的区别

视图动画系统仅提供为 `View` (/reference/android/view/View) 对象添加动画效果的功能，因此，如果您想为非 对象添加动画效果，则必须实现自己的代码才能做到。视图动画系统也存在一些限制，因为它仅公开 对象的部分方面来供您添加动画效果；例如，您可以对视图的缩放和旋转添加动画效果，但无法对背景颜色这样做。

视图动画系统的另一个缺点是它只会在绘制视图的位置进行修改，而不会修改实际的视图本身。例如，如果您为某个按钮添加了动画效果，使其可以在屏幕上移动，该按钮会正确绘制，但能够点击按钮的实际位置并不会更改，因此您必须通过实现自己的逻辑来处理此事件。

有了属性动画系统，您就可以完全摆脱这些束缚，还可以为任何对象（视图和非视图）的任何属性添加动画效果，并且实际修改的是对象本身。属性动画系统在执行动画方面也更为强健。概括地讲，您可以为要添加动画效果的属性（例如颜色、位置或大小）分配 Animator，还可以定义动画的各个方面，例如多个 Animator 的插值和同步。

不过，视图动画系统的设置需要的时间较短，需要编写的代码也较少。如果视图动画可以完成您需要执行的所有操作，或者现有代码已按照您需要的方式运行，则无需使用属性动画系统。在某些用例中，也可以针对不同的情况同时使用这两种动画系统。

API 概览

您可以在 `android.animation` (/reference/android/animation/package-summary) 中找到属性动画系统的大多数 API。视图动画系统已经在 `android.view.animation` (/reference/android/view/animation/package-summary) 中定义了许多插值器，因此，您也可以在属性动画系统中使用这些插值器。下表描述了属性动画系统的主要组成部分。

`Animator` (/reference/android/animation/Animator) 类提供了创建动画的基本结构。您通常不会直接使用此类，因为它只提供极少的功能，这些功能必须经过扩展才能全面支持为值添加动画效果。以下子类可扩展 `Animator` (/reference/android/animation/Animator)：

表 1. Animator

类	说明
`ValueAnimator` (/reference/android/animation/ValueAnimator)	属性动画的主计时引擎，它也可计算要添加动画效果的属性的值。它具有计算动画值所需的所有核心功能，同时包含每个动画的计时详情、有关动画是否重复播放的信息、用于接收更新事件的监听器以及设置待评估自定义类型的功能。为属性添加动画效果分为两个步骤：计算添加动画效果之后的值，以及对要添加动画效果的对象和属性设置这些值。`ValueAnimator` (/reference/android/animation/ValueAnimator) 不会执行第二个步骤，因此，您必须监听由 `ValueAnimator` (/reference/android/animation/ValueAnimator) 计算的值的更新情况，并使用您自己的逻辑修改要添加动画效果的对象。如需了解详情，请参阅使用 ValueAnimator 添加动画效果 (#value-animator)部分。
`ObjectAnimator` (/reference/android/animation/ObjectAnimator)	`ValueAnimator` (/reference/android/animation/ValueAnimator) 的子类，用于设置目标对象和对象属性以添加动画效果。此类会在计算出动画的新值后相应地更新属性。在大多数情况下，您不妨使用 `ObjectAnimator` (/reference/android/animation/ObjectAnimator)，因为它可以极大地简化对目标对象的值添加动画效果这一过程。不过，有时您需要直接使用 `ValueAnimator` (/reference/android/animation/ValueAnimator)，因为 `ObjectAnimator` (/reference/android/animation/ObjectAnimator) 存在其他一些限制，例如要求目标对象具有特定的访问器方法。
`AnimatorSet` (/reference/android/animation/AnimatorSet)	此类提供一种将动画分组在一起的机制，以使它们彼此相对运行。您可以将动画设置为一起播放、按顺序播放或者在指定的延迟时间后播放。如需了解详情，请参阅使用 AnimatorSet 编排多个动画 (#choreography)部分。

评估程序负责告知属性动画系统如何计算指定属性的值。它们使用由 `Animator` (/reference/android/animation/Animator) 类提供的计时数据（即动画的起始值和结束值），并根据这些数据计算属性添加动画效果之后的值。属性动画系统可提供以下评估程序：

表 2. 评估程序

类/接口	说明
`IntEvaluator` (/reference/android/animation/IntEvaluator)	这是用于计算 `int` 属性的值的默认评估程序。
`FloatEvaluator` (/reference/android/animation/FloatEvaluator)	这是用于计算 `float` 属性的值的默认评估程序。
`ArgbEvaluator` (/reference/android/animation/ArgbEvaluator)	这是用于计算颜色属性的值（用十六进制值表示）的默认评估程序。
`TypeEvaluator` (/reference/android/animation/TypeEvaluator)	此接口用于创建您自己的评估程序。如果您要添加动画效果的对象属性不是 `int`、`float` 或颜色，那么您必须实现 `TypeEvaluator` (/reference/android/animation/TypeEvaluator) 接口，才能指定如何计算对象属性添加动画效果之后的值。如果您想以不同于默认行为的方式处理 `int`、`float`和颜色，您还可以为这些类型的值指定自定义 `TypeEvaluator` (/reference/android/animation/TypeEvaluator)。如需详细了解如何编写自定义评估程序，请参阅使用 TypeEvaluator (#type-evaluator) 部分。

时间插值器指定了如何根据时间计算动画中的特定值。例如，您可以指定动画在整个动画中以线性方式播放，即动画在整个播放期间匀速移动；也可以指定动画使用非线性时间，例如动画在开始后加速并在结束前减速。表 3 介绍了 `android.view.animation` (/reference/android/view/animation/package-summary) 中包含的插值器。如果下表提供的插值器都不能满足您的需求，请实现 `TimeInterpolator` (/reference/android/animation/TimeInterpolator) 接口并创建您自己的插值器。如需详细了解如何编写自定义插值器，请参阅使用插值器 (#interpolators)。

表 3. 插值器

类/接口	说明
`AccelerateDecelerateInterpolator` (/reference/android/view/animation/AccelerateDecelerateInterpolator)	该插值器的变化率在开始和结束时缓慢但在中间会加快。
`AccelerateInterpolator` (/reference/android/view/animation/AccelerateInterpolator)	该插值器的变化率在开始时较为缓慢，然后会加快。
`AnticipateInterpolator` (/reference/android/view/animation/AnticipateInterpolator)	该插值器先反向变化，然后再急速正向变化。
`AnticipateOvershootInterpolator` (/reference/android/view/animation/AnticipateOvershootInterpolator)	该插值器先反向变化，再急速正向变化，然后超过定位值，最后返回到最终值。
`BounceInterpolator` (/reference/android/view/animation/BounceInterpolator)	该插值器的变化会跳过结尾处。
`CycleInterpolator` (/reference/android/view/animation/CycleInterpolator)	该插值器的动画会在指定数量的周期内重复。
`DecelerateInterpolator` (/reference/android/view/animation/DecelerateInterpolator)	该插值器的变化率开始很快，然后减速。

LinearInterpolator (/reference/android/view/animation/LinearInterpolator)	该插值器的变化率恒定不变。
OvershootInterpolator (/reference/android/view/animation/OvershootInterpolator)	该插值器会急速正向变化，再超出最终值，然后返回。
TimeInterpolator (/reference/android/animation/TimeInterpolator)	该接口用于实现您自己的插值器。

使用 ValueAnimator 添加动画效果

借助 ValueAnimator (/reference/android/animation/ValueAnimator) 类，您可以为动画播放期间某些类型的值添加动画效果，只需指定一组要添加动画效果的 int、float 或颜色值即可。您可以通过调用 ValueAnimator (/reference/android/animation/ValueAnimator) 的任一工厂方法来获取它：ofInt() (/reference/android/animation/ValueAnimator#ofInt(int...))、ofFloat() (/reference/android/animation/ValueAnimator#ofFloat(float...)) 或 ofObject() (/reference/android/animation/ValueAnimator#ofObject(android.animation.TypeEvaluator, java.lang.Object...))。例如：

KOTLIN (#KOTLIN)JAVA

```
ValueAnimator animation = ValueAnimator.ofFloat(0f, 100f);
animation.setDuration(1000);
animation.start();
```

在上述代码中，当 start() (/reference/android/animation/ValueAnimator#start()) 方法运行时，ValueAnimator (/reference/android/animation/ValueAnimator) 会开始计算 1000ms 时长内 0 和 100 之间的动画的值。

您还可以通过执行以下操作来指定要添加动画效果的自定义类型：

KOTLIN (#KOTLIN)JAVA

```
ValueAnimator animation = ValueAnimator.ofObject(new MyTypeEvaluator(), startPropertyValue, endPropertyValue);
animation.setDuration(1000);
animation.start();
```

在上述代码中，当 start() (/reference/android/animation/ValueAnimator#start()) 方法运行时，ValueAnimator (/reference/android/animation/ValueAnimator) 会开始使用 MyTypeEvaluator 提供的逻辑计算 1000ms 时长内 startPropertyValue 和 endPropertyValue 之间的动画的值。

您可以通过向 ValueAnimator (/reference/android/animation/ValueAnimator) 对象添加 AnimatorUpdateListener (/reference/android/animation/ValueAnimator.AnimatorUpdateListener) 来使用动画的值，如以下代码所示：

KOTLIN (#KOTLIN)JAVA

```
animation.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
    @Override
    public void onAnimationUpdate(ValueAnimator updatedAnimation) {
        // You can use the animated value in a property that uses the
        // same type as the animation. In this case, you can use the
        // float value in the translationX property.
        float animatedValue = (float)updatedAnimation.getAnimatedValue();
        textView.setTranslationX(animatedValue);
    }
});
```

在 onAnimationUpdate() (/reference/android/animation/ValueAnimator.AnimatorUpdateListener#onAnimationUpdate(android.animation.ValueAnimator)) 方法中，您可以访问更新后的动画值，并将其用在您的某个视图的属性中。如需详细了解监听器，请参阅动画监听器 (#listeners)部分。

使用 ObjectAnimator 添加动画效果

ObjectAnimator (/reference/android/animation/ObjectAnimator) 是上一部分中讨论的 ValueAnimator (/reference/android/animation/ValueAnimator) 的子类，它融合了 ValueAnimator (/reference/android/animation/ValueAnimator) 的计时引擎和值计算以及为目标对象的命名属性添加动画效果这一功能。这可以极大地简化为任何对象添加动画效果的过程，因为动画属性会自动更新，因此您也无需再实现 ValueAnimator.AnimatorUpdateListener (/reference/android/animation/ValueAnimator.AnimatorUpdateListener) 了。

实例化 ObjectAnimator (/reference/android/animation/ObjectAnimator) 与 ValueAnimator (/reference/android/animation/ValueAnimator) 的过程类似，但您也可以指定对象和该对象属性的名称（以字符串形式），以及要在哪些值之间添加动画效果：

KOTLIN (#KOTLIN)JAVA

```
ObjectAnimator animation = ObjectAnimator.ofFloat(textView, "translationX", 100f);
animation.setDuration(1000);
animation.start();
```

要使 ObjectAnimator (/reference/android/animation/ObjectAnimator) 正确更新属性，您必须执行以下操作：

- 要添加动画效果的对象属性必须具有 `set<PropertyName>()` 形式的 setter 函数（采用驼峰式大小写形式）。由于 ObjectAnimator (/reference/android/animation/ObjectAnimator) 会在动画过程中自动更新属性，它必须能够使用此 setter 方法访问该属性。例如，如果属性名称为 `foo`，则需要使用 `setFoo()` 方法。如果此 setter 方法不存在，您有三个选择：
  - 如果您有权限，可将 setter 方法添加到类中。
  - 使用您有权更改的封装容器类，让该封装容器使用有效的 setter 方法接收值并将其转发给原始对象。
  - 改用 ValueAnimator (/reference/android/animation/ValueAnimator)。
- 如果您在 ObjectAnimator (/reference/android/animation/ObjectAnimator) 的一个工厂方法中仅为 `values...` 参数指定了一个值，则系统会假定它是动画的结束值。因此，要添加动画效果的对象属性必须具有用于获取动画起始值的 getter 函数。getter 函数必须采用 `get<PropertyName>()` 形式。例如，如果属性名称为 `foo`，则需要使用 `getFoo()` 方法。
- 要添加动画效果的属性的 getter（如果需要）和 setter 方法的操作对象必须与您为 ObjectAnimator (/reference/android/animation/ObjectAnimator) 指定的起始值和结束值的类型相同。例如，如果构建以下 ObjectAnimator (/reference/android/animation/ObjectAnimator)，则必须具有 `targetObject.setPropName(float)` 和 `targetObject.getPropName(float)`：

  ```
  ObjectAnimator.ofFloat(targetObject, "propName", 1f)
  ```

- 根据您要添加动画效果的属性或对象，您可能需要对视图调用 invalidate() (/reference/android/view/View#invalidate()) 方法，以强制屏幕使用添加动画效果之后的值重新绘制自身。您可以在 onAnimationUpdate() (/reference/android/animation/ValueAnimator.AnimatorUpdateListener#onAnimationUpdate(android.animation.ValueAnimator)) 回调中执行此操作。例如，如果为可绘制对象的颜色属性添加动画效果，则仅当该对象重新绘制自身时，屏幕才会刷新。视图的所有属性 setter（如 setAlpha() (/reference/android/view/View#setAlpha(float)) 和 setTranslationX() (/reference/android/view/View#setTranslationX(float))）都会使视图失效，因此，在使用新值调用这些方法时，您无需使视图失效。如需详细了解监听器，请参阅动画监听器 (#listeners)部分。

使用 AnimatorSet 编排多个动画

在许多情况下，您需要根据一个动画开始或结束的时间来播放另一个动画。借助 Android 系统，您可以将动画捆绑到一个 AnimatorSet (/reference/android/animation/AnimatorSet) 中，以便指定是同时播放动画、按顺序播放还是在指定的延迟时间后播放。您还可以相互嵌套 AnimatorSet (/reference/android/animation/AnimatorSet) 对象。

以下代码段通过以下方式播放相应的 Animator (/reference/android/animation/Animator) 对象：

1. 播放 `bounceAnim`。
2. 同时播放 `squashAnim1`、`squashAnim2`、`stretchAnim1` 和 `stretchAnim2`。
3. 播放 `bounceBackAnim`。
4. 播放 `fadeAnim`。

KOTLIN (#KOTLIN) JAVA

```
AnimatorSet bouncer = new AnimatorSet();
bouncer.play(bounceAnim).before(squashAnim1);
bouncer.play(squashAnim1).with(squashAnim2);
bouncer.play(squashAnim1).with(stretchAnim1);
bouncer.play(squashAnim1).with(stretchAnim2);
bouncer.play(bounceBackAnim).after(stretchAnim2);
ValueAnimator fadeAnim = ObjectAnimator.ofFloat(newBall, "alpha", 1f, 0f);
fadeAnim.setDuration(250);
AnimatorSet animatorSet = new AnimatorSet();
animatorSet.play(bouncer).before(fadeAnim);
animatorSet.start();
```

动画监听器

您可以使用下述监听器来监听动画播放期间的重要事件。

- `Animator.AnimatorListener` (/reference/android/animation/Animator.AnimatorListener)
  - `onAnimationStart()` (/reference/android/animation/Animator.AnimatorListener#onAnimationStart(android.animation.Animator)) - 在动画开始播放时调用。
  - `onAnimationEnd()` (/reference/android/animation/Animator.AnimatorListener#onAnimationEnd(android.animation.Animator)) - 在动画结束播放时调用。
  - `onAnimationRepeat()` (/reference/android/animation/Animator.AnimatorListener#onAnimationRepeat(android.animation.Animator)) - 在动画重复播放时调用。
  - `onAnimationCancel()` (/reference/android/animation/Animator.AnimatorListener#onAnimationCancel(android.animation.Animator)) - 在动画取消播放时调用。取消的动画也会调用 `onAnimationEnd()` (/reference/android/animation/Animator.AnimatorListener#onAnimationEnd(android.animation.Animator))，无论它们以何种方式结束。
- `ValueAnimator.AnimatorUpdateListener` (/reference/android/animation/ValueAnimator.AnimatorUpdateListener)
  - `onAnimationUpdate()` (/reference/android/animation/ValueAnimator.AnimatorUpdateListener#onAnimationUpdate(android.animation.ValueAnimator)) - 对动画的每一帧调用。监听此事件即可使用 `ValueAnimator` (/reference/android/animation/ValueAnimator) 在动画播放期间生成的计算值。要使用该值，请查询传递到事件中的 `ValueAnimator` (/reference/android/animation/ValueAnimator) 对象，以使用 `getAnimatedValue()` (/reference/android/animation/ValueAnimator#getAnimatedValue()) 方法获取当前添加动画效果之后的值。如果使用了 `ValueAnimator` (/reference/android/animation/ValueAnimator)，则必须实现此监听器。

    根据您要添加动画效果的属性或对象，您可能需要对视图调用 `invalidate()` (/reference/android/view/View#invalidate())，以强制屏幕上的相应区域使用添加动画效果之后的新值重新绘制自身。例如，如果为可绘制对象的颜色属性添加动画效果，则仅当该对象重新绘制自身时，屏幕才会刷新。视图的所有属性 setter（如 `setAlpha()` (/reference/android/view/View#setAlpha(float)) 和 `setTranslationX()` (/reference/android/view/View#setTranslationX(float))）都会使视图失效，因此，在使用新值调用这些方法时，您无需使视图失效。

如果您不一定需要实现 `Animator.AnimatorListener` (/reference/android/animation/Animator.AnimatorListener) 接口的所有方法，则可以扩展 `AnimatorListenerAdapter` (/reference/android/animation/AnimatorListenerAdapter) 类，而非实现 接口。`AnimatorListenerAdapter` (/reference/android/animation/AnimatorListenerAdapter) 类提供了方法的空实现，可供您选择替换。

例如，以下代码段可仅为 `onAnimationEnd()` (/reference/android/animation/Animator.AnimatorListener#onAnimationEnd(android.animation.Animator)) 回调创建 `AnimatorListenerAdapter` (/reference/android/animation/AnimatorListenerAdapter)：

KOTLIN (#KOTLIN) JAVA

```
ValueAnimator fadeAnim = ObjectAnimator.ofFloat(newBall, "alpha", 1f, 0f);
fadeAnim.setDuration(250);
fadeAnim.addListener(new AnimatorListenerAdapter() {
public void onAnimationEnd(Animator animation) {
    balls.remove(((ObjectAnimator)animation).getTarget());
```

```
    }
```

为 ViewGroup 对象的布局更改添加动画效果

属性动画系统提供对 ViewGroup 对象的更改添加动画效果的功能，还可轻松为视图对象本身添加动画效果。

您可使用 LayoutTransition (/reference/android/animation/LayoutTransition) 类为 ViewGroup 内的布局更改添加动画效果。当您向 ViewGroup 添加视图或删除其中的视图时，或当您使用 VISIBLE (/reference/android/view/View#VISIBLE)、INVISIBLE (/reference/android/view/View#INVISIBLE) 或 GONE (/reference/android/view/View#GONE) 调用视图的 setVisibility(.) (/reference/android/view/View#setVisibility(int)) 方法时，这些视图可能会经历出现和消失动画。向 ViewGroup 添加视图或删除其中的视图时，其中剩余的视图还可能以动画形式移动到新位置。您可以调用 setAnimator(.) (/reference/android/animation/LayoutTransition#setAnimator(int, android.animation.Animator)) 并使用以下任一 LayoutTransition (/reference/android/animation/LayoutTransition) 常量传入 Animator (/reference/android/animation/Animator) 对象，从而在 LayoutTransition (/reference/android/animation/LayoutTransition) 对象中定义相应动画：

- APPEARING - 该标记表示动画在容器中出现的项上运行。
- CHANGE_APPEARING - 该标记表示动画在因某个新项目在容器中出现而变化的项上运行。
- DISAPPEARING - 该标记表示动画在从容器中消失的项上运行。
- CHANGE_DISAPPEARING - 该标记表示动画在因某个项从容器中消失而变化的项上运行。

您可以为这四类事件定义您自己的自定义动画，从而自定义布局转换的外观，或者告诉动画系统使用默认动画。

API 演示中的 LayoutAnimations (/resources/samples/ApiDemos/src/com/example/android/apis/animation/LayoutAnimations) 示例介绍了如何为布局转换定义动画，然后在要添加动画效果的视图对象上设置动画。

LayoutAnimationsByDefault (/resources/samples/ApiDemos/src/com/example/android/apis/animation/LayoutAnimationsByDefault) 及其对应的 layout_animations_by_default.xml (/resources/samples/ApiDemos/res/layout/layout_animations_by_default) 布局资源文件介绍了如何在 XML 中为 ViewGroup 启用默认布局转换。您唯一需要做的就是将 ViewGroup 的 `android:animateLayoutchanges` 属性设置为 `true`。例如：

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:id="@+id/verticalContainer"
    android:animateLayoutChanges="true" />
```

将此属性设置为 true 可自动为添加到 ViewGroup 或从中删除的视图以及该 ViewGroup 中剩余的视图添加动画效果。

使用 StateListAnimator 为视图状态更改添加动画效果

通过 StateListAnimator (/reference/android/animation/StateListAnimator) 类，您可以定义在视图状态更改时运行的 Animator。此对象充当 Animator (/reference/android/animation/Animator) 对象的封装容器，只要指定的视图状态（例如“按下”或“聚焦”）发生更改，就会调用该动画。

可使用根 `<selector>` 元素和子 `<item>` 元素在 XML 资源中定义 StateListAnimator (/reference/android/animation/StateListAnimator)，每个元素都指定一个由 StateListAnimator (/reference/android/animation/StateListAnimator) 类定义的不同视图状态。每个 `<item>` 都包含一个属性动画集 (/guide/topics/resources/animation-resource#Property)的定义。

例如，以下文件创建了一个状态列表 Animator，可在按下后更改视图的 x 和 y 比例：

res/xml/animate_scale.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- the pressed state; increase x and y size to 150% -->
    <item android:state_pressed="true">
```

```
    <set>
        <objectAnimator android:propertyName="scaleX"
            android:duration="@android:integer/config_shortAnimTime"
            android:valueTo="1.5"
            android:valueType="floatType"/>
        <objectAnimator android:propertyName="scaleY"
            android:duration="@android:integer/config_shortAnimTime"
            android:valueTo="1.5"
            android:valueType="floatType"/>
    </set>
</item>
<!-- the default, non-pressed state; set x and y size to 100% -->
<item android:state_pressed="false">
    <set>
        <objectAnimator android:propertyName="scaleX"
            android:duration="@android:integer/config_shortAnimTime"
            android:valueTo="1"
            android:valueType="floatType"/>
        <objectAnimator android:propertyName="scaleY"
            android:duration="@android:integer/config_shortAnimTime"
            android:valueTo="1"
            android:valueType="floatType"/>
    </set>
</item>
</selector>
```

要将状态列表 Animator 附加到视图，请添加 `android:stateListAnimator` (/reference/android/view/View#attr_android:stateListAnimator) 属性，如下所示：

```
<Button android:stateListAnimator="@xml/animate_scale"
        ... />
```

现在，当此按钮的状态发生变化时，会使用 `animate_scale.xml` 中定义的动画。

或者，如果要转为在代码中将状态列表 Animator 分配给视图，则可使用 `AnimatorInflater.loadStateListAnimator()` (/reference/android/animation/AnimatorInflater#loadStateListAnimator(android.content.Context, int)) 方法，然后使用 `View.setStateListAnimator()` (/reference/android/view/View#setStateListAnimator(android.animation.StateListAnimator)) 方法将 Animator 分配给相应视图。

您还可以使用 `AnimatedStateListDrawable` (/reference/android/graphics/drawable/AnimatedStateListDrawable) 在状态更改间播放可绘制动画，而不是为视图的属性添加动画效果。Android 5.0 中的一些系统微件默认会使用这些动画。以下示例展示了如何将 `AnimatedStateListDrawable` (/reference/android/graphics/drawable/AnimatedStateListDrawable) 定义为 XML 资源：

```
<!-- res/drawable/myanimstatedrawable.xml -->
<animated-selector
    xmlns:android="http://schemas.android.com/apk/res/android">

    <!-- provide a different drawable for each state-->
    <item android:id="@+id/pressed" android:drawable="@drawable/drawableP"
        android:state_pressed="true"/>
    <item android:id="@+id/focused" android:drawable="@drawable/drawableF"
        android:state_focused="true"/>
    <item android:id="@id/default"
        android:drawable="@drawable/drawableD"/>

    <!-- specify a transition -->
    <transition android:fromId="@+id/default" android:toId="@+id/pressed">
        <animation-list>
            <item android:duration="15" android:drawable="@drawable/dt1"/>
            <item android:duration="15" android:drawable="@drawable/dt2"/>
            ...
        </animation-list>
    </transition>
    ...
```

```
    </animated-selector>
```

使用 TypeEvaluator

如果要为 Android 系统无法识别的类型添加动画效果，则可以通过实现 `TypeEvaluator` (/reference/android/animation/TypeEvaluator) 接口来创建您自己的评估程序。Android 系统可以识别的类型为 `int`、`float` 或颜色，分别由 `IntEvaluator` (/reference/android/animation/IntEvaluator)、`FloatEvaluator` (/reference/android/animation/FloatEvaluator) 和 `ArgbEvaluator` (/reference/android/animation/ArgbEvaluator) 类型评估程序提供支持。

`TypeEvaluator` (/reference/android/animation/TypeEvaluator) 接口中只有一种要实现的方法，那就是 `evaluate()` (/reference/android/animation/TypeEvaluator#evaluate(float, T, T)) 方法。这样，您使用的 Animator 就会在动画的当前点为添加动画效果之后的属性返回适当的值。`FloatEvaluator` (/reference/android/animation/FloatEvaluator) 类演示了如何做到这一点：

KOTLIN (#KOTLIN)**JAVA**

```
public class FloatEvaluator implements TypeEvaluator {

    public Object evaluate(float fraction, Object startValue, Object endValue) {
        float startFloat = ((Number) startValue).floatValue();
        return startFloat + fraction * (((Number) endValue).floatValue() - startFloat);
    }
}
```

注意：当 `ValueAnimator` (/reference/android/animation/ValueAnimator)（或 `ObjectAnimator` (/reference/android/animation/ObjectAnimator)）运行时，它会计算动画当前的已完成动画分数（一个介于 0 和 1 之间的值），然后根据您正在使用的插值器来计算该分数的插值版本。插值分数是 `TypeEvaluator` (/reference/android/animation/TypeEvaluator) 通过 `fraction` 参数接收的，因此您在计算添加动画效果之后的值时无需考虑插值器。

使用插值器

插值器指定了如何根据时间计算动画中的特定值。例如，您可以指定动画在整个动画中以线性方式播放，即动画在整个播放期间匀速移动；也可以指定动画使用非线性时间，例如动画在开始后加速并在结束前减速。

动画系统中的插值器会接收来自 Animator 的分数，该分数表示动画的已播放时间。插值器会修改此分数，使其与要提供的动画类型保持一致。Android 系统在 `android.view.animation package` (/reference/android/view/animation/package-summary) 中提供了一组常用的插值器。如果这些插值器都不能满足您的需求，您可以实现 `TimeInterpolator` (/reference/android/animation/TimeInterpolator) 接口并创建您自己的插值器。

以下示例对比了默认插值器 `AccelerateDecelerateInterpolator` (/reference/android/view/animation/AccelerateDecelerateInterpolator) 和 `LinearInterpolator` (/reference/android/view/animation/LinearInterpolator) 计算插值分数的方式。`LinearInterpolator` (/reference/android/view/animation/LinearInterpolator) 对已完成动画分数没有任何影响。`AccelerateDecelerateInterpolator` (/reference/android/view/animation/AccelerateDecelerateInterpolator) 会在动画开始后加速，并在动画结束前减速。以下方法定义了这些插值器的逻辑：

AccelerateDecelerateInterpolator

KOTLIN (#KOTLIN)**JAVA**

```
@Override
public float getInterpolation(float input) {
    return (float)(Math.cos((input + 1) * Math.PI) / 2.0f) + 0.5f;
}
```

LinearInterpolator

KOTLIN (#KOTLIN)**JAVA**

```
@Override
public float getInterpolation(float input) {
    return input;
}
```

下表表示这些插值器为时长 1000ms 的动画计算的近似值：

已完成毫秒数	已完成动画分数/插值分数（线性）	插值分数（加速/减速）
0	0	0
200	0.2	0.1
400	0.4	0.345
600	0.6	0.8
800	0.8	0.9
1000	1	1

如上表所示，`LinearInterpolator` (/reference/android/view/animation/LinearInterpolator) 以相同的速度更改值，每 200ms 变化 0.2。`AccelerateDecelerateInterpolator` (/reference/android/view/animation/AccelerateDecelerateInterpolator) 在 200ms 到 600ms 之间更改值的速度会快于 `LinearInterpolator` (/reference/android/view/animation/LinearInterpolator)，在 600ms 到 1000ms 之间会慢一些。

指定关键帧

`Keyframe` (/reference/android/animation/Keyframe) 对象由时间值对组成，用于在动画的特定时间定义特定的状态。每个关键帧还可以用自己的插值器控制动画在上一关键帧时间和此关键帧时间之间的时间间隔内的行为。

要实例化 `Keyframe` (/reference/android/animation/Keyframe) 对象，您必须使用它的任一工厂方法（`ofInt()` (/reference/android/animation/Keyframe#ofInt(float))、`ofFloat()` (/reference/android/animation/Keyframe#ofFloat(float)) 或 `ofObject()` (/reference/android/animation/Keyframe#ofObject(float))）来获取类型合适的 。然后，通过调用 `ofKeyframe()` (/reference/android/animation/PropertyValuesHolder#ofKeyframe(android.util.Property, android.animation.Keyframe...)) 工厂方法来获取 `PropertyValuesHolder` (/reference/android/animation/PropertyValuesHolder) 对象。获取对象后，您可以通过传入 `PropertyValuesHolder` (/reference/android/animation/PropertyValuesHolder) 对象以及要添加动画效果的对象来获取 Animator。以下代码段演示了如何做到这一点：

KOTLIN (#KOTLIN)**JAVA**

```
Keyframe kf0 = Keyframe.ofFloat(0f, 0f);
Keyframe kf1 = Keyframe.ofFloat(.5f, 360f);
Keyframe kf2 = Keyframe.ofFloat(1f, 0f);
PropertyValuesHolder pvhRotation = PropertyValuesHolder.ofKeyframe("rotation", kf0, kf1, kf2);
ObjectAnimator rotationAnim = ObjectAnimator.ofPropertyValuesHolder(target, pvhRotation);
rotationAnim.setDuration(5000);
```

为视图添加动画效果

属性动画系统支持为视图对象添加经过简化的动画，与视图动画系统相比，它具有一定的优势。视图动画系统通过更改绘制视图对象的方式来转换视图对象。这是在每个视图的容器中处理的，因为视图本身没有可操控的属性。这会导致视图在表面上添加了动画效果，但视图对象本身没有任何变化。这会产生不好的效果，例如，某个对象已经在屏幕的其他位置绘制，但它仍位于其原始位置。在 Android 3.0 中，我们添加了新的属性以及相应的 getter 和 setter 方法来消除此缺陷。

属性动画系统可以通过更改视图对象中的实际属性来为屏幕上的视图添加动画效果。此外，当视图的属性发生更改时，视图还会自动调用 `invalidate()` (/reference/android/view/View#invalidate()) 方法来刷新屏幕。`View` (/reference/android/view/View) 类中有利于属性动画的新属性包

括：

- translationX 和 translationY：这些属性用于控制视图所在的位置，值为视图的布局容器所设置的左侧坐标和顶部坐标的增量。
- rotation、rotationX 和 rotationY：这些属性用于控制视图围绕轴心点进行的 2D（属性）和 3D 旋转。
- scaleX 和 scaleY：这些属性用于控制视图围绕其轴心点进行的 2D 缩放。
- pivotX 和 pivotY：这些属性用于控制旋转和缩放转换所围绕的轴心点的位置。默认情况下，轴心点位于对象的中心。
- x 和 y：这些是简单的实用属性，用于描述视图在容器中的最终位置，值分别为左侧值与 translationX 值的和以及顶部值与 translationY 值的和。
- alpha：表示视图的 Alpha 透明度。此值默认为 1（不透明），值为 0 则表示完全透明（不可见）。

要为视图对象的属性（例如其颜色或旋转值）添加动画效果，您只需创建一个属性 Animator 并指定要添加动画效果的视图属性即可。例如：

KOTLIN (#KOTLIN)JAVA

```
ObjectAnimator.ofFloat(myView, "rotation", 0f, 360f);
```

如需详细了解如何创建 Animator，请参阅介绍如何使用 ValueAnimator (#value-animator) 和 ObjectAnimator (#object-animator) 添加动画效果的部分。

使用 ViewPropertyAnimator 添加动画效果

ViewPropertyAnimator (/reference/android/view/ViewPropertyAnimator) 有助于使用单个底层 Animator (/reference/android/animation/Animator) 对象轻松为 View (/reference/android/view/View) 的多个属性并行添加动画效果。它的行为方式与 ObjectAnimator (/reference/android/animation/ObjectAnimator) 非常相似，因为它会修改视图属性的实际值，但在同时为多个属性添加动画效果时，它更为高效。此外，使用 ViewPropertyAnimator (/reference/android/view/ViewPropertyAnimator) 的代码更加简洁，也更易读。以下代码段展示了在同时为视图的 x 和 y 属性添加动画效果时，使用多个 ObjectAnimator (/reference/android/animation/ObjectAnimator) 对象、使用单个 ObjectAnimator (/reference/android/animation/ObjectAnimator) 对象以及使用 ViewPropertyAnimator (/reference/android/view/ViewPropertyAnimator) 的区别。

多个 ObjectAnimator 对象

KOTLIN (#KOTLIN)JAVA

```
ObjectAnimator animX = ObjectAnimator.ofFloat(myView, "x", 50f);
ObjectAnimator animY = ObjectAnimator.ofFloat(myView, "y", 100f);
AnimatorSet animSetXY = new AnimatorSet();
animSetXY.playTogether(animX, animY);
animSetXY.start();
```

一个 ObjectAnimator

KOTLIN (#KOTLIN)JAVA

```
PropertyValuesHolder pvhX = PropertyValuesHolder.ofFloat("x", 50f);
PropertyValuesHolder pvhY = PropertyValuesHolder.ofFloat("y", 100f);
ObjectAnimator.ofPropertyValuesHolder(myView, pvhX, pvhY).start();
```

ViewPropertyAnimator

KOTLIN (#KOTLIN)JAVA

```
myView.animate().x(50f).y(100f);
```

如需详细了解 ViewPropertyAnimator (/reference/android/view/ViewPropertyAnimator)，请参阅相应的 Android 开发者博文 (http://android-developers.blogspot.com/2011/05/introducing-viewpropertyanimator.html)。

在 XML 中声明动画

属性动画系统支持您使用 XML 声明属性动画，而不是以编程方式进行声明。通过在 XML 中定义动画，您可以轻松地在多个 Activity 中重复使用动画，还能更轻松地修改动画序列。

为了将使用新属性动画 API 的动画文件与使用旧版视图动画 (/guide/topics/graphics/view-animation)框架的动画文件区分开来，从 Android 3.1 开始，您应将属性动画的 XML 文件保存到 `res/animator/` 目录中。

以下属性动画类具有带相应 XML 标签的 XML 声明支持：

- ValueAnimator (/reference/android/animation/ValueAnimator) - `<animator>`
- ObjectAnimator (/reference/android/animation/ObjectAnimator) - `<objectAnimator>`
- AnimatorSet (/reference/android/animation/AnimatorSet) - `<set>`

要查找 XML 声明中可使用的属性，请参阅动画资源 (/guide/topics/resources/animation-resource#Property)。以下示例依次播放两组对象动画，其中第一个嵌套集会同时播放两个对象动画：

```
<set android:ordering="sequentially">
    <set>
        <objectAnimator
            android:propertyName="x"
            android:duration="500"
            android:valueTo="400"
            android:valueType="intType"/>
        <objectAnimator
            android:propertyName="y"
            android:duration="500"
            android:valueTo="300"
            android:valueType="intType"/>
    </set>
    <objectAnimator
        android:propertyName="alpha"
        android:duration="500"
        android:valueTo="1f"/>
</set>
```

为了运行此动画，您必须将代码中的 XML 资源扩充为 AnimatorSet (/reference/android/animation/AnimatorSet) 对象，然后在开始动画集之前为所有动画设置目标对象。调用 setTarget() (/reference/android/animation/AnimatorSet#setTarget(java.lang.Object)) 即可为 AnimatorSet (/reference/android/animation/AnimatorSet) 的所有子项设置一个目标对象，这是系统提供的便捷方式。以下代码展示了如何做到这一点：

KOTLIN (#KOTLIN)JAVA

```
AnimatorSet set = (AnimatorSet) AnimatorInflater.loadAnimator(myContext,
    R.animator.property_animator);
set.setTarget(myObject);
set.start();
```

您还可以在 XML 中声明 ValueAnimator (/reference/android/animation/ValueAnimator)，如以下示例所示：

```
<animator xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="1000"
    android:valueType="floatType"
    android:valueFrom="0f"
    android:valueTo="-100f" />
```

要使用代码中的上一个 ValueAnimator (/reference/android/animation/ValueAnimator)，您必须扩充对象、添加 AnimatorUpdateListener (/reference/android/animation/ValueAnimator.AnimatorUpdateListener)、获取更新后的动画值并在某个视图的属性中使用它，如下面的代码所示：

KOTLIN (#KOTLIN)**JAVA**

```
ValueAnimator xmlAnimator = (ValueAnimator) AnimatorInflater.loadAnimator(this,
        R.animator.animator);
xmlAnimator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
    @Override
    public void onAnimationUpdate(ValueAnimator updatedAnimation) {
        float animatedValue = (float)updatedAnimation.getAnimatedValue();
        textView.setTranslationX(animatedValue);
    }
});

xmlAnimator.start();
```

如需了解定义属性动画的 XML 语法，请参阅动画资源 (/guide/topics/resources/animation-resource#Property)。

对界面性能的潜在影响

用于更新界面的 Animator 会使动画运行的每一帧都进行额外的渲染。因此，使用资源密集型动画可能会对应用的性能产生负面影响。

为界面添加动画效果所需的工作已添加到渲染管道的动画阶段 (/topic/performance/rendering/profile-gpu#at)。您可以启用 **GPU 渲染模式分析**并监控动画阶段，以了解您的动画是否会影响应用的性能。如需了解详情，请参阅 GPU 渲染模式分析演示 (/studio/profile/dev-options-rendering)。