

# HenCoder Plus 讲义

## 属性动画和硬件加速

### 属性动画

#### ViewPropertyAnimator

返回 `Animate` 對象。

使用 `View.animate()` 创建对象，以及使用 `ViewPropertyAnimator.translationX()` 等方法来设置动画；

可以连续调用来设置多个动画；

可以用 `setDuration()` 来设置持续时间；

可以用 `setStartDelay()` 来设置开始延时；

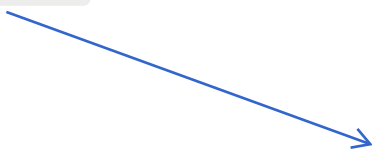
以及其他一些便捷方法。

#### ObjectAnimator

使用 `ObjectAnimator.ofXxx()` 来创建对象，以及使用 `ObjectAnimator.start()` 来主动启动动画。它的优势在于，可以为自定义属性设置动画。

```
ObjectAnimator animator = ObjectAnimator.ofObject(view, "radius", Utils.dp2px(200));
```

另外，自定义属性需要设置 `getter` 和 `setter` 方法，并且 `setter` 方法里需要调用 `invalidate()` 来触发重绘：



於性能考慮，當屬性改變時，畫面並不會自動刷新。

```

public float getRadius() {
    return radius;
}

public void setRadius(float radius) {
    this.radius = radius;
    invalidate();
}

```

可以使用 `setDuration()` 来设置持续时间；

可以用 `setStartDelay()` 来设置开始延时；

以及其他一些便捷方法。

## Interpolator

插值器，用于设置时间完成度到动画完成度的计算公式，直白地说即设置动画的速度曲线，通过 `setInterpolator(Interpolator)` 方法来设置。

常用的有 `AccelerateDecelerateInterpolator`

→ 不涉及元件的出場入場。

`AccelerateInterpolator`

`DecelerateInterpolator`

→ 減速動畫，常用於入場。

`LinearInterpolator`。

→ 加速動畫，常用於出場，像是飛出螢幕。

等速動畫。

## PropertyValuesHolder

用于设置更加详细的动画，例如多个属性应用于同一个对象：

```

PropertyValuesHolder holder1 =
    PropertyValuesHolder.ofFloat("radius",
        Utils.dp2px(200));
PropertyValuesHolder holder2 =
    PropertyValuesHolder.ofFloat("offset",
        Utils.dp2px(100));
ObjectAnimator animator =
    PropertyValuesHolder.ofPropertyValuesHolder(view,
        holder1, holder2);

```

或者，配合使用 `Keyframe`，对一个属性分多个段：

```
Keyframe keyframe1 = Keyframe.ofFloat(0,
    Utils.dpToPixel(100));
Keyframe keyframe2 = Keyframe.ofFloat(0.5f,
    Utils.dpToPixel(250));
Keyframe keyframe3 = Keyframe.ofFloat(1,
    Utils.dpToPixel(200));
PropertyValuesHolder holder =
    PropertyValuesHolder.ofKeyframe("radius", keyframe1,
    keyframe2, keyframe3);
ObjectAnimator animator =
    ObjectAnimator.ofPropertyValuesHolder(view, holder);
```

## AnimatorSet

将多个 Animator 合并在一起使用，先后顺序或并列顺序都可以：

```
AnimatorSet animatorSet = new AnimatorSet();
    animatorSet.playTogether(animator1, animator2);
    animatorSet.start();
```

## TypeEvaluator

時間完成度對於動畫完成度的映射。

用于设置动画完成度到属性具体值的计算公式。默认的 `ofInt()` `ofFloat()` 已经有了自带的 `IntEvaluator` `FloatEvaluator`，但有的时候需要自己设置 `Evaluator`。例如，对于颜色，需要为 int 类型的颜色设置 `ArgbEvaluator`，而不是让它们使用 `IntEvaluator`：

```
animator.setEvaluator(new ArgbEvaluator());
```

如果你对 `ArgbEvaluator` 的效果不满意，也可以自己写一个 `HsvEvaluator`：

```

public class HsvEvaluator implements
TypeEvaluator<Integer> {
    @Override
    public Object evaluate(float fraction, Object
startValue, Object endValue) {
        ...
    }
}

```

另外，对于不支持的类型，也可以使用 `ofObject()` 来在创建 Animator 的同时就设置上 Evaluator，比如 `NameEvaluator`：

```

public class NameEvaluator implements
TypeEvaluator<String> {
    List<String> names = ...;

    @Override
    public String evaluate(float fraction, String
startValue, String endValue) {
        if (!names.contains(startValue)) {
            throw new IllegalArgumentException("Start
value not existed");
        }
        if (!names.contains(endValue)) {
            throw new IllegalArgumentException("End
value not existed");
        }
        int index = (int) ((names.indexOf(endValue) -
names.indexOf(startValue)) * fraction);
        return names.get(index);
    }
}

ObjectAnimator animator = ObjectAnimator.ofObject(view,
"name", new NameEvaluator(), "Jack");

```

## Listeners

和 View 的点击、长按监听器一样，Animator 也可以使用 `setXxxListener()` `addXxxListener()` 来设置监听器。

## ValueAnimator

这是最基本的 Animator，它不和具体的某个对象联动，而是直接对两个数值进行渐变计算。使用很少。

## 硬件加速

### 硬件加速是什么

使用 CPU 繪製就稱為軟件繪製。

- 使用 CPU 绘制到 Bitmap，然后把 Bitmap 贴到屏幕，就是软件绘制；
- 使用 CPU 把绘制内容转换成 GPU 操作，交给 GPU，由 GPU 负责真正的绘制，就叫硬件绘制；
- 使用 GPU 绘制就叫做硬件加速

使用 GPU 繪製就稱為硬件繪製。

### 怎么就加速了？

- GPU 分摊了工作
- GPU 绘制简单图形（例如方形、圆形、直线）在硬件设计上具有先天优势，会更快
- 流程得到优化（重绘流程涉及的内容更少）

### 硬件加速的缺陷：

若使用軟件繪製，當重繪時，對 CPU 而言，就是一整塊黏合的圖，而對 GPU 而言，它依然擁有圖層，與 CPU 不同，它僅許改動圖層而不用整個重繪。

兼容性。由于使用 GPU 的绘制（暂时）无法完成某些绘制，因此对于一些特定的 API，需要关闭硬件加速来转回到使用 CPU 进行绘制。

### 离屏缓冲：

是一個緩存的概念。

- 离屏缓冲是什么：单独的一个绘制 View（或 View 的一部分）的区域
- `setLayerType()` 和 `saveLayer()`
  - `setLayerType()` 是对整个 View，不能针对 `onDraw()` 里面的某一具体过程
    - 这个方法常用来关闭硬件加速，但它的定位和定义都不只是一个「硬件加速开关」。它的作用是为绘制设置一个离屏缓冲，让后面的绘制都单独写在这个离屏缓冲内。如果参数填写 `LAYER_TYPE_SOFTWARE`，

会把离屏缓冲设置为一个 Bitmap，即使用软件绘制来进行缓冲，这样就导致在设置离屏缓冲的同时，将硬件加速关闭了。但需要知道，这个方法被用来关闭硬件加速，只是因为 Android 并没有提供一个便捷的方法在 View 级别简单地开关硬件加速而已。

- saveLayer() 是针对 Canvas 的，所以在 onDraw() 里可以使用 saveLayer() 来圈出具体哪部分绘制要用离屏缓冲

- 然而.....最新的文档表示这个方法太重了，能不用就别用，尽量用 setLayerType() 代替

→ saveLayer 是一次性的離屏緩衝，實際上，它仍有它 setLayer 無法取代的地方。

備註：Android 可以在 AndroidManifest 開啟或關閉硬件加速的功能，預設為開啟，但該功能是針對 Android 全局；而 Android 並沒有提供特定 View 關閉硬件加速的功能，因此，有人會藉由「setLayerType(LAYER\_TYPE\_SOFTWARE, ...)」來實現關閉特定 View 的硬件加速功能，但我們必須清楚的是，setLayerType() 是設定離屏緩衝的功能，而不是開關硬件加速的方法，僅是剛好具有這樣的作用。

## 问题和建议？

课上技术相关的问题，都可以去群里和大家讨论，对于比较通用的、有价值的问题，可以去我们的知识星球提问。

具体技术之外的问题和建议，都可以找丢物线（微信：diuwuxian），丢丢会为你解答技术以外的一切。



# 觉得好？

如果你觉得课程很棒，欢迎给我们好评呀！<https://ke.qq.com/comment/index.html?cid=381952>

一定要是你真的觉得好，再给我们好评。不要仅仅因为对扔物线的支持而好评（报名课程已经是你最大的支持了，再不够的话 B 站多来点三连我也很开心），另外我们也坚决不做好评返现等任何的交易。我们只希望，在课程对你有帮助的前提下，可以看到你温暖的评价。

## 更多内容：

- 网站：<https://hencoder.com>；<https://kaixue.io>
- 各大搜索引擎、微信公众号、微博、知乎、掘金、哔哩哔哩、YouTube、西瓜视频、抖音、快手、微视：统一账号「扔物线」，我会持续输出优质的技术内容，欢迎大家关注。
- 哔哩哔哩快捷传送门：<https://space.bilibili.com/27559447>

大家如果喜欢我们的课程，还请去扔物线的哔哩哔哩，帮我素质三连，感谢大家！