# Android动画

* Animation框架：

实现原理是每次绘制视图时View所在的ViewGroup中的drawChild函数获取该View的Animation的Transformation值，然后调用canvas.concat(transformToApply.getMatrix())，通过矩阵运算完成动画帧，如果动画没有完成，继续调用invalidate()函数，启动下次绘制来驱动动画，动画过程中的帧之间间隙时间是绘制函数所消耗的时间，可能会导致动画消耗比较多的CPU资源。

* Animator框架：Animator Set和ObjectAnimator配合

在Animator框架中使用最多的是AnimatorSet和ObjectAnimator配合，使用ObjectAnimator进行更精细化控制（PropertyValuesHolder），只控制一个对象的一个属性值，多个ObjectAnimator组合到AnimatorSet形成一个动画。而且ObjectAnimator能够自动驱动，可以调用setFrameDelay(longframeDelay)设置动画帧之间的间隙时间，调整帧率，减少动画过程中频繁绘制界面，而在不影响动画效果的前提下减少CPU资源消耗。

# Tween动画（补间动画）

View animation，它只是改变了View对象绘制的位置，而没有改变View对象本身。也就是说View大小位置不变，造成所占据的空间还是固定的，只是在既定空间上显示动画。

官方文档有XML形式。

|  |
| --- |
| <objectAnimator xmlns:android=*"http://schemas.android.com/apk/res/android"*  android:duration=*"1000"*  android:valueTo=*"200"*  android:valueType=*"floatType"*  android:propertyName=*"y"*  android:repeatCount=*"1"*  android:repeatMode=*"reverse"*/> |
| Java调用代码 |
| **ImageView spaceshipImage = (ImageView)findViewById(R.id.spaceshipImage); Animation hyperspaceJumpAnimation=AnimationUtils.loadAnimation(this, R.anim.hyperspace\_jump); spaceshipImage.startAnimation(hyperspaceJumpAnimation);** |

# Drawable帧动画

多个图片逐帧播放，类似幻灯片

|  |
| --- |
| <animation-list xmlns:android="http://schemas.android.com/apk/res/android"  android:oneshot="true">  <item android:drawable="@drawable/rocket\_thrust1" android:duration="200" />  <item android:drawable="@drawable/rocket\_thrust2" android:duration="200" />  <item android:drawable="@drawable/rocket\_thrust3" android:duration="200" />  </animation-list> |

# 属性动画

基本认知

* 针对一个view对象在屏幕上进行动画显示。
* 属性动画，指的是该动画显示过程中可以修改view的属性，包括大小，位置，透明度。
* 可以实现的效果是，移动位置，三轴旋转，透明度变化，缩放。

相关属性

* Duration：动画的持续时间
* TimeInterpolation：属性值的计算方式，如先快后慢。

**API：setInterpolator(TimeInterpolator value)**

* TypeEvaluator：根据属性的开始、结束值与TimeInterpolation计算出的因子计算出当前时间的属性值

可以自定义TypeEvaluator，

* Repeat Count and behavoir：重复次数与方式，如播放3次、5次、无限循环，可以此动画一直重复，或播放完时再反向播放

**API：setRepeatCount(int value)和setRepeatMode(int value)**

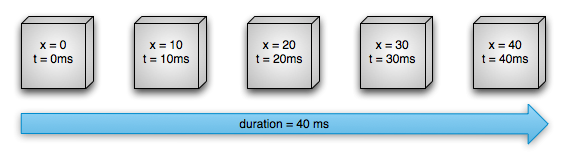
* Animation sets：动画集合，即可以同时对一个对象应用几个动画，这些动画可以同时播放也可以对不同动画设置不同开始偏移

**API:playTogether，playSequentially**

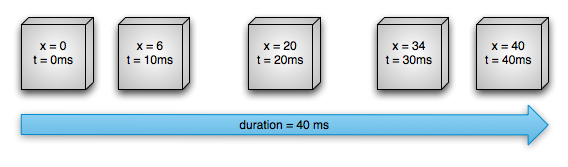
* Frame refreash delay：多少时间刷新一次，即每隔多少时间计算一次属性值，默认为10ms，最终刷新时间还受系统进程调度与硬件的影响

**API：setFrameDelay(long frameDelay)**

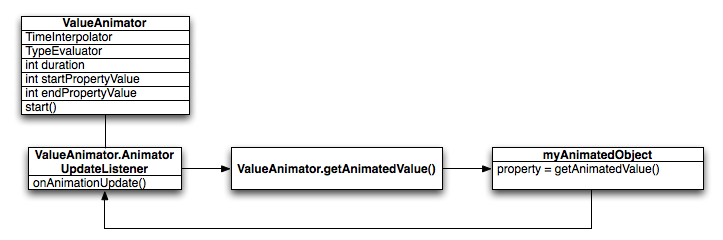
对于下图的动画，这个对象的X坐标在40ms内从0移动到40 pixel.按默认的10ms刷新一次，这个对象会移动4次，每次移动40/4=10pixel。



也可以改变属性值的改变方法，即设置不同的interpolation，在下图中运动速度先逐渐增大再逐渐减小



下图显示了与上述动画相关的关键对象



ValueAnimator  表示一个动画，包含动画的开始值，结束值，持续时间等属性。

ValueAnimator封装了一个TimeInterpolator，TimeInterpolator定义了属性值在开始值与结束值之间的插值方法。

ValueAnimator还封装了一个TypeAnimator，根据开始、结束值与TimeIniterpolator计算得到的值计算出属性值。

ValueAnimator根据动画已进行的时间跟动画总时间（duration）的比计算出一个时间因子（0~1），然后根据TimeInterpolator计算出另一个因子，最后TypeAnimator通过这个因子计算出属性值

涉及关键类

|  |  |  |  |
| --- | --- | --- | --- |
| [java.lang.Object](https://developer.android.com/reference/java/lang/Object.html) | | | |
| ↳ | [android.animation.Animator](https://developer.android.com/reference/android/animation/Animator.html) | | |
|  | ↳ | [android.animation.ValueAnimator](https://developer.android.com/reference/android/animation/ValueAnimator.html) | |
|  |  | ↳ | android.animation.**ObjectAnimator** |
| [java.lang.Object](https://developer.android.com/reference/java/lang/Object.html) | | | |
| ↳ | [android.animation.Animator](https://developer.android.com/reference/android/animation/Animator.html) | | |
|  | ↳ | android.animation.**AnimatorSet** | |
| [java.lang.Object](https://developer.android.com/reference/java/lang/Object.html) | | | |
| ↳ | android.animation.**Keyframe** | | |
| [java.lang.Object](https://developer.android.com/reference/java/lang/Object.html) | | | |
| ↳ | android.animation.**PropertyValuesHolder** | | |

# 简单例子

例子1

单一操作，使用ObjectAnimator类，使view360度旋转

参数：**ObjectAnimator ofFloat (Object target, String propertyName, float... values)**

1. View 对象
2. 要操作的属性名，具体见下文
3. 多参数：属性变化过程的指定数值
4. setDuration动画时长

|  |
| --- |
| public void viewRotationAnimator(){  //一个简单的动画，非复合动画，直接用ObjectAnimator及可start  ObjectAnimator.*ofFloat*(MyView, "rotation", 0F, 360F).setDuration(200).start();//360度旋转  } |

例子2 AnimatorSet 异步操作

复合操作，利用ObjectAnimator和AnimatorSet分别沿着x轴y轴方向移动50px。

|  |
| --- |
| public void viewXYAnimator(){  ObjectAnimator animX = ObjectAnimator.*ofFloat*(text1, "x", 0f,50f);  ObjectAnimator animY = ObjectAnimator.*ofFloat*(text1, "y", 0f,50f);  AnimatorSet animSetXY = new AnimatorSet();  animSetXY.playTogether(animX, animY);// playTogether表示异步叠加，还有playSequentially，表示同步执行  animSetXY.setDuration(100);  animSetXY.start();  } |

例子3 AnimatorSet 同步操作

复合操作，利用ObjectAnimator和AnimatorSet同步机制，先移动再缩放。

|  |
| --- |
| public void setSyncTextAnimator(){  ObjectAnimator animX = ObjectAnimator.*ofFloat*(text\_sync, "x", 0f,200f);  ObjectAnimator animscaleY = ObjectAnimator.*ofFloat*(text\_sync, "scaleY", 1f,2f,1f);  AnimatorSet animSetXY = new AnimatorSet();  animSetXY.playSequentially(animX, animscaleY);// playTogether表示异步叠加，还有playSequentially，表示同步执行  animSetXY.setDuration(2000);  animSetXY.start();  } |

总结

AnimatorSet方法：同步或异步的执行一系列方法。可以先定义一个Animator的List，然后用如下 方法执行。

|  |  |
| --- | --- |
| 同步执行list | playSequentially(List<Animator> items) |
| 同步执行，多参数 | playSequentially(Animator... items) |
| 异步执行集合 | playTogether(Collection<Animator> items) |
| 异步执行，多参数 | playTogether(Animator... items) |

例子4 PropertyValuesHolder和keyFrame的使用

**利用PropertyValuesHolder存储多属性，利用Keyframe细化帧。**

**PropertyValuesHolder包括一系列静态方法，用于存储动画属性，返回一个PropertyValuesHolder对象。**

**keyFrame是一个 时间/值 对，通过它可以定义一个在特定时间的特定状态，即关键帧，而且在两个keyFrame之间可以定义不同的Interpolator，就好像多个动画的拼接，第一个动画的结束点是第二个动画的开始点。KeyFrame是抽象类，要通过ofInt(),ofFloat(),ofObject()获得适当的KeyFrame，然后通过PropertyValuesHolder.ofKeyframe获得PropertyValuesHolder对象，如以下例子：**

|  |  |
| --- | --- |
| Keyframe静态方法，参数百分比，位置值 | [ofFloat](https://developer.android.com/reference/android/animation/Keyframe.html#ofFloat(float, float))(float fraction, float value) |
| PropertyValuesHolder静态方法 | [ofKeyframe](https://developer.android.com/reference/android/animation/PropertyValuesHolder.html#ofKeyframe(java.lang.String, android.animation.Keyframe...))([String](https://developer.android.com/reference/java/lang/String.html) propertyName, [Keyframe...](https://developer.android.com/reference/android/animation/Keyframe.html) values) |
| ObjectAnimator静态方法 | [ofPropertyValuesHolder](https://developer.android.com/reference/android/animation/ObjectAnimator.html#ofPropertyValuesHolder(java.lang.Object, android.animation.PropertyValuesHolder...))([Object](https://developer.android.com/reference/java/lang/Object.html) target, [PropertyValuesHolder...](https://developer.android.com/reference/android/animation/PropertyValuesHolder.html) values) |

|  |
| --- |
| public void viewPropertyValuesHolder(){  Keyframe kf0 = Keyframe.*ofFloat*(0f, 0f);  Keyframe kf1 = Keyframe.*ofFloat*(0.5f, 180f);  Keyframe kf2 = Keyframe.*ofFloat*(1f, 360f);  // 用三个关键帧构造PropertyValuesHolder对象，最后装配到ObjectAnimator  PropertyValuesHolder pvhX = PropertyValuesHolder.*ofKeyframe*(  "rotation", kf0, kf1, kf2);  PropertyValuesHolder moveX = PropertyValuesHolder.*ofFloat*("translationX",0f,200f,300f,0f);  PropertyValuesHolder scaleX = PropertyValuesHolder.*ofFloat*("scaleX",1.0f,0.2f,1f);  //ofPropertyValuesHolder是多参数的，每个PropertyValuesHolder相当于动画预设值，这些动画异步执行  ObjectAnimator animator= ObjectAnimator.*ofPropertyValuesHolder*(text3, moveX).setDuration(3000);  animator.start();  } |

例子5 利用addUpdateListener控制view

动画目标：一个图片逐渐显示出来，且整个view逐渐变大，不会留有空白位置。

步骤：

1. 新建Image，设置背景资源
2. 在**UNSPECIFIED模式下调用Measure方法**计算view实际大小
3. 设置动画属性是translationY从负位置移动到0位置。
4. 调用**addUpdateListener**方法监听
5. 在绘制每一帧时，重新设置view的位置，并重绘

|  |
| --- |
| **public** **void** setImageState(){  image1=**new** ImageView(**this**);  image1.setBackgroundResource(R.drawable.*sample\_1*);  LinearLayout.LayoutParams layoutParams= **new** LinearLayout.LayoutParams(LinearLayout.LayoutParams.*WRAP\_CONTENT*,LinearLayout.LayoutParams.*WRAP\_CONTENT*);  image1.setLayoutParams(layoutParams);  layout\_image.addView(image1);  //测量方案，先利用UNSPECIFIED模式测量View实际需要的空间大小。  **int** childHeightSpec = MeasureSpec.*makeMeasureSpec*(0, MeasureSpec.*UNSPECIFIED*);  **int** childWidthSpec = MeasureSpec.*makeMeasureSpec*(0, MeasureSpec.*UNSPECIFIED*);  image1.measure(childWidthSpec, childHeightSpec);  **final** **int** measuredHeight=-image1.getMeasuredHeight();  Log.*i*(TAG, "measuredHeight"+measuredHeight);  //目标效果是从上向下，逐步显示出图片，也就是左上角坐标从一个负位置到0位置，负位置大小就是MeasureHeight  PropertyValuesHolder moveX = PropertyValuesHolder.*ofFloat*("translationY",measuredHeight,0f);  ObjectAnimator animator= ObjectAnimator.*ofPropertyValuesHolder*(image1,moveX).setDuration(3000);  //该方法用来监听动画绘制过程中的每一帧的改变，通过这个方法，我们可以在动画重绘的过程中，实现自己的逻辑。  animator.addUpdateListener(**new** AnimatorUpdateListener() {  @Override  **public** **void** onAnimationUpdate(ValueAnimator animation) {  // **TODO** Auto-generated method stub  **float** i=1.0f;  //该方法会获取每一帧的“插值”，也就是结束值与开始值之间按照时间区间分割出多个值。  //例如，移动时从Y轴方向view从-200移动到0，会添加-200到0的很多帧代表的数值：-199，-198等等直到0  **float** ll=(Float) animation.getAnimatedValue();  LinearLayout.LayoutParams params = **new** LinearLayout.LayoutParams(300,(**int**) (ll-measuredHeight));  layout\_image.setLayoutParams(params);  image1.setLayoutParams(params);  //重绘自身的位置，构成View占用空间由小变大的效果。  image1.requestLayout();  }  });  animator.start();  } |

例子6 动画加速，减速

Time interplator定义了属性值变化的方式，如线性均匀改变，开始慢然后逐渐快等。在Property Animation中是TimeInterplator，在View Animation中是Interplator，这两个是一样的，在3.0之前只有Interplator，3.0之后实现代码转移至了TimeInterplator。Interplator继承自TimeInterplator，内部没有任何其他代码。

* AccelerateInterpolator　　　　　     加速，开始时慢中间加速
* DecelerateInterpolator　　　 　　   减速，开始时快然后减速
* AccelerateDecelerateInterolator　   先加速后减速，开始结束时慢，中间加速
* AnticipateInterpolator　　　　　　  反向 ，先向相反方向改变一段再加速播放
* AnticipateOvershootInterpolator　   反向加回弹，先向相反方向改变，再加速播放，会超出目的值然后缓慢移动至目的值
* BounceInterpolator　　　　　　　  跳跃，快到目的值时值会跳跃，如目的值100，后面的值可能依次为85，77，70，80，90，100
* CycleIinterpolator　　　　　　　　 循环，动画循环一定次数，值的改变为一正弦函数：Math.sin(2 \* mCycles \* Math.PI \* input)
* LinearInterpolator　　　　　　　　 线性，线性均匀改变
* OvershottInterpolator　　　　　　  回弹，最后超出目的值然后缓慢改变到目的值
* TimeInterpolator　　　　　　　　   一个接口，允许你自定义interpolator，以上几个都是实现了这个接口

|  |
| --- |
| animator.setInterpolator(**new** AccelerateInterpolator(**this**, **null**));  animator.start(); |

## 例子7自定义TypeEvaluator

# 属性值

参数单位

移动单位是px，也即是像素，旋转的角度是0到350度，缩放的比例是0到1倍。

* translationX and translationY: These properties control where the View is located as a delta from its left and top coordinates which are set by its layout container.
* rotation, rotationX, and rotationY: These properties control the rotation in 2D (rotation property) and 3D around the pivot point.
* scaleX and scaleY: These properties control the 2D scaling of a View around its pivot point.
* pivotX and pivotY: These properties control the location of the pivot point, around which the rotation and scaling transforms occur. By default, the pivot point is located at the center of the object.
* x and y: These are simple utility properties to describe the final location of the View in its container, as a sum of the left and top values and translationX and translationY values.
* alpha: Represents the alpha transparency on the View. This value is 1 (opaque) by default, with a value of 0 representing full transparency (not visible).

|  |  |
| --- | --- |
| **translationX**  **translationY** | 在自身的Layout中，距离左上角的x方向，y方向的距离。  动画效果是从原位置移动到设定的位置。 |
| **Rotation**  **rotationX rotationY** | 旋转，Rotation是绕着Z轴旋转，也就是垂直屏幕的轴。  RotationX，RotationY分别是绕着X和Y轴。 |
| **scaleX**  **scaleY** | 围绕X或Y轴缩放，2D显示控制尺寸大小  ObjectAnimator.ofFloat(myview, "scaleX", 1f,0.5f,1f);//缩小到0.5倍再还原 |
| **pivotX**  **pivotY** | 旋转或缩放的中心轴的轴点位置。 |
| **x**  **y** | 在自身的Layout中，距离左上角的x方向，y方向的实际距离。也就是padding值+ **translationX 或translationY值** |
| **alpha** | 透明度，1表示默认显示不透明，0表示全透明，看不见 |