Animation

Animation에는 Frame Animation과 Tween Animation 두 종류가 있다. Frame Animation은 전통적인 방법으로 매 프레임마다 다른 이미지를 노출시켜 Animation 효과를 얻는 것이다. Tween Animation은 변화시킬 속성의 from, to를 명시해서 중간 값을 계산하고 Animation 효과를 얻는 것이다.

API 11 이전의 **View Animation**은 이 두가지를 이용했다. API 11부터 **Property Animation**이라는 새로운 Animation이 등장하였으며, API 14 부터는 더욱 향상된 **ViewPropertyAnimation**이 새로 소개되었다.

**1. Property Animation**

기존의 android.view.animation 패키지를 이용해서 애니매이션을 구현하는 것이 가능했다. 하지만 기존의 애니매이션에는 몇가지 단점이 있다. 먼저, View 오브젝트만을 애니매이션 할 수 있었다. 예를 들어, Button, TextView, LinearLayout 등을 애니매이션 하는 것이 가능했다. 하지만 Drawable을 변화시키는 것이 불가능했다. 그리고, 애니매이션하는 속성에 한계가 있었다. rote, scale, move는 가능하지만 색을 변화시키는 것은 불가능했다. 마지막으로 기존의 animation은 눈에 보이는 속성만 animation되었다. 예를 들어 버튼을 움직이면 눈에 보이는 버튼의 움직임은 정상적이지만, 버튼의 클릭 이벤트가 일어나는 위치는 개발자가 따로 코딩을 해줘야 했다. 컨테이너 안에서의 물리적인 위치는 변하지 않았다. 이런 문제점 때문에 Property Animation이 생겨났다.

**1.1 API Overview**

**1.1.1 Animator**

Animator는 새로운 animation클래스의 수퍼클래스로 일반적인 속성과 메서드를 포함한다. 서브클래스는 ValueAnimator, AnimatorSet이 있다. ValueAnimator는 시스템의 중요한 Timing 엔진이다. AnimatorSet은 여러 애니매이션이 동시에 진행되도록 한다.

애니매이션의 LifeCycle 이벤트를 처리하기 위해서 AnimatorListener 인터페이스를 이용해 Animator에 리스너를 추가한다.

anim.addListener(new Animator.AnimatorListener() {

        public void onAnimationStart(Animator animation) {}

        public void onAnimationEnd(Animator animation) {

          // do something when the animation is done

        }

        public void onAnimationCancel(Animator animation) {}

        public void onAnimationRepeat(Animator animation) {}

    });

편리를 위해 AnimatorListenerAdapter 클래스를 이용할 수도 있다. 메서드중 일부만 구현하고 싶다면 Adapter를 이용한다.

anim.addListener(new AnimatorListenerAdapter() {

        public void onAnimationEnd(Animator animation) {

          // do something when the animation is done

        }

    });

**1.1.2 ValueAnimator**

메인 타이밍 엔진이다. 재생되는 Proeprty 값을 계산한다. 애니매이션 값들을 계산하는 핵심 기능을 포함한다. 그리고 각 애니매이션의 timing detail을 포함한다. (애니매이션이 반복되었는지 아닌지와 같은 정보, 업데이트 이벤트를 처리하는 리스너, 커스텀 타입)

ObjectAnimator는 ValueAnimator의 서브 클래스로 target object에 계산된 값을 쉽게 적용할 수 있도록 한다. 언제 새로운 값을 계산했는지에 맞춰 property를 갱신한다. target object에서 재생값을 처리하는 것이 쉽기 때문에 이 클래스를 주로 사용하게 되겠지만, 몇가지 제한이 있어 ValueAnimator를 직접 사용할 수도 있다. 제한 중에 하나는 target object가 setter()를 가져야 한다는 것이다. 아래는 ValueAnimator를 직접 사용하는 예이다.

ValueAnimator anim = ValueAnimator.ofFloat(0f, 1f);

    anim.setDuration(500);

  anim.start();

만약 매 프레임마다 animated value로 어떤 작업을 하고 싶다면 AnimatorUpdateListener()를 이용한다. AnimatorUpdateListner()는 매 프레임마다 동작한다.

anim.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {

        public void onAnimationUpdate(ValueAnimator animation) {

            Float value = (Float) animation.getAnimatedValue();

            // do something with value...

        }

    });

float가 아닌 다른 구조를 변화시키고 싶다면 TypeEvaluator를 이용한다. 시스템은 int, float밖에 처리하지 않는데, TypeEvaluator는 그 외의 구조를 어떻게 int, float로 치환시켜 계산하는지를 시스템에 알려준다. 포인트를 변화시키는 애니매이션이라면 ValueAnimator의 ofobject()메서드에 Evaluator를 매개변수로 추가해 주면 된다.

Point p0 = new Point(0, 0);

    Point p1 = new Point(100, 200);

   ValueAnimator anim = ValueAnimator.ofObject(pointEvaluator, p0, p1);

**1.1.3 ObjectAnimator**

ObjectAnimator는 간단하게 애니매이션을 만들고, 계산된 값을 target object에 적용한다. myObject라는 오브젝트를 fade out 시키는 예는 다음과 같다.

ObjectAnimator.ofFloat(myObject, "alpha", 0f).start();

이 예에서 세번째 매개변수 0f 앞에 1를 추가한다면 (myObject, "alpha", 1f, 0f), 1을 from, 0을 to로 설정하여 animate 한다. 시작값을 명시해 주지 않으면 현재값을 시작값으로 하여 animate 한다. XML 리소스로 정의 할 수도 있다.

<objectAnimator xmlns:android="http://schemas.android.com/apk/res/android"

        android:valueTo="0"

        android:propertyName="alpha"/>

XML에서는 애니매이션만 정의하고 target object는 지정할 수 없다. 코드에서 지정해야 한다.

ObjectAnimator anim = AnimatorInflator.loadAnimator(context, resID);

    anim.setTarget(myObject);

    anim.start();

ObjectAnimation은 target object의 settter 메서드를 이용해 값을 적용시키고, getter 메서드를 이용해 현재 animated value를 얻는다. 따라서 이 예에서 target object인 myObject는 public 속성의 setter, getter 메서드를 갖는다.

public void setAlpha(float value);

 public float getAlpha();

만약 이 두 메서드를 가지지 않는다면, 애니매이션은 runtime에 fail된다.

**1.1.4 AnimatorSet**

AnimatorSet 클래스는 기존의 AnimationSet과 유사하지만 더 확장되었다. 다음 예제를 보면 이해가 쉽다. v1을 fadeout 시키고, v2를 fadein 하면서 slide 시킨다. (play, with, after의 사용)

ObjectAnimator fadeOut = ObjectAnimator.ofFloat(v1, "alpha", 0f);

ObjectAnimator mover = ObjectAnimator.ofFloat(v2, "translationX", -500f, 0f);

ObjectAnimator fadeIn = ObjectAnimator.ofFloat(v2, "alpha", 0f, 1f);

AnimatorSet animSet = new AnimatorSet().play(mover).with(fadeIn).after(fadeOut);;

animSet.start();

AnimatorSet도 Xml로 정의할 수 있다.

**1.1.5 TypeEvaluator**

Property Animation은 커스텀 타입을 animate 시킬 수 있다고 했다. 시스템은 float, int밖에 계산하지 못하므로 float, int가 아닌 프로퍼티를 어떻게 float로 계산하는지에 대한 방법을 시스템에 알려줘야 한다. 이때 TypeEvaluator 인터페이스를 상속하여 Evaluator를 만들어야 한다.

구현할 메서드는 evaluate() 하나밖에 없다. 위에 Point를 변화시키는 예제가 있었다.

Point p0 = new Point(0, 0);

Point p1 = new Point(100, 200);

ValueAnimator anim = ValueAnimator.ofObject(pointEvaluator, p0, p1);

매개변수로 사용된 pointEvaluator는 다음 PointEvaluator의 인스턴스이다.

public class PointEvaluator implements TypeEvaluator {

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

            Point startPoint = (Point) startValue;

            Point endPoint = (Point) endValue;

            return new Point(startPoint.x + fraction \* (endPoint.x - startPoint.x),

                startPoint.y + fraction \* (endPoint.y - startPoint.y));

        }

    }

위의 예와 같이, TypeEvlauator를 상속받아 PointEvaluator를 만들고 evaluate()를 구현한 다음, 인스턴스를 Animator에 함꼐 넣어주면 int, float가 아닌 값도 animate 시킬 수 있다.

기본적으로 제공되는 Evuator는 IntEvaluator, FloatEvaluator, ArgbEvaluator(색)이 있다.

**1.1.6 추가된 View Properties**

Property Animation은 Object가 그려지는 모습 뿐 아니라 실제 위치도 변경된다고 했는데, 이를 위해 View에 추가된 속성들이 있다.

- translationX and traslationY : 이 두 속성은 뷰의 위치를 컨트롤하는데, 레이아웃에 설정된 left, top에서부터의 delta값으로 컨트롤한다. 이 두 값을 이용해 View를 translate 시키고 싶으면, ObjectAnimator.ofFloat(view, "translationX", 0f, 100f);.라고 쓰면 된다.

- rotation, rotationx and rotationY : 이 속성들은 pivot point를 기준으로 2D, 3D 회전을 컨트롤한다.

- scaleX and scaleY : 이 속성들은 2D scale을 컨트롤한다.

-pivotX, pivotY: pivot point의 위치를 컨트롤한다. 디폴트값은 오브젝트의 center이다.

- x and y : 뷰의 최종 위치를 표시한다. left/top 과 traslationX/translationY의 합이다.

- alpha: 뷰의 알파값이다. 1이 디폴트이며 fadeout 시키고 싶으면 ObjectAnimator.ofFloat(view, "alpha", 0f); 로 서술한다.

알아야 할것은 이 속성들이 set/get 메서드에서 유용하다는 점이다. set 메서드는 적절하게 invalidation 할 수 있게 해주기 때문에, 각 값에 변경이 있을 때 Animation system에 접근하여 올바른 동작을 할 수 있도록 한다.

**2. ViewPropertyAnimation**

PropertyAnimation에는 몇 가지 개선사항이 있다. 첫번째, property를 명시할 때 (예를 들어 ‘alpha’) ‘alpha’라는 문자열은 setAlpha()메서드로 변하게 된다. 이 동작을 JNI를 통해 하게 되는데 여기에는 오버헤드가 존재한다. 두 번째, Animator 그 자신이다. 각 Animator는 동일한 timing machanism을 공유하는데도 각 property마다 해당 task가 존재한다. 마지막으로, property들은 이전의 값과, 현재의 값을 모두 invalidate한다. 여러 property가 병렬적으로 변경된다면 오버헤드가 존재하게 된다.

PropertyAnimation은 여러 property를 하나의 Animator 안에서 동시에 animate할 수 있게 한다. 또한 값을 계산하는 것과 target view에 적용하고 invalide 하는 일을 손쉽게 해준다.

myView를 fadeout 하는 예는 다음과 같다.

before : **ObjectAnimator.ofFloat(myView, "alpha", 0f).start();a**

after : **myView.animate().alpha(0);**

ViewPropertyAnimation에서 Animator는 View가 만든다. ainmate() 메서드는 Animator를 리턴하여 애니매이션을 수행하도록 한다. Animator를 생성하는 코드가 필요하지 않고, 매개변수도 간단해 졌으며 start() 메서드를 더이상 호출하지 않아도 된다. 더욱 효율적이다.

또한 여러가지 property를 한꺼번에 animate하는 일도 더욱 간단해 졌다.

**myView.animate().x(500).y(500);**