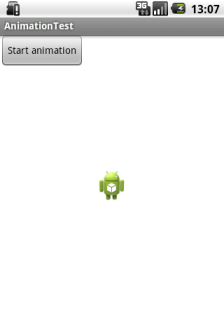
**[Анимация видов (view)](http://ru-code-android.livejournal.com/5392.html" \t "_self)**



В этой статье рассмотрим как сделать анимацию элементов интерфейса в android. Под элементами интерфейса в данном случае имеются в виду все наследники класса View (полный список наследников можно посмотреть в [документации класса View](http://developer.android.com/reference/android/view/View.html%C2%BB)). Анимация это простой способ сделать приложение более живеньким :)   
  
**1.** Начнем с создания тестового полигона. Сделаем простое приложение с кнопкой и картинкой посередине экрана. Код приводить не буду, он простой, если что, смотрите в исходниках (они в конце статьи).   
  
  
**2.** В директории /res/anim создадим файл anim.xml и напишем туда

<?**xml** version="**1.0**" encoding="utf-8"?>

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

android:shareInterpolator="false">

<alpha

android:fromAlpha="0.0"

android:toAlpha="1.0"

android:duration="1000"/>

</set>

Это описание анимации, которое мы будем применять к нашей картинке. Подробнее что тут происходит рассмотрим ниже, а пока просто скопируем это в файл.  
  
**3.** Чтобы загрузить анимацию из xml файла используется статический метод класса [AnimationUtils](http://developer.android.com/reference/android/view/animation/AnimationUtils.html" \t "_self)  
**loadAnimation(Context context, int id)**, где *context* — текущий контекст, а *id* — идентификатор ресурса с анимацией. Метод возвращает экземпляр класса Animation.  
[Animation](http://developer.android.com/reference/android/view/animation/Animation.html) — абстрактный класса для представления анимации в приложении.   
Чтобы применить ее, полученный экземпляр класса Animation передается методу  
**startAnimation(Animation animation)** класса View (и всех его наследников).  
  
**4.** Напишем в файл AnimationTestActivity.java:

**public** **class** AnimationTestActivity **extends** Activity {

ImageView image;

**Button** button;

Animation anim;

@Override

**protected** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.main);

image = (ImageView)findViewById(R.id.image);

button = (**Button**)findViewById(R.id.button);

anim = AnimationUtils.loadAnimation(**this**, R.anim.anim); // 1

button.setOnClickListener(**new** OnClickListener() {

@Override

**public** **void** onClick(View v) {

image.startAnimation(anim); //2

}

});

}

}

1) Читаем файл с идентификатором R.anim.anim (что соответствует файлу /res/anim/anim.xml) и получаем экземпляр класса Animation.  
2) По нажатию на кнопку применяем анимацию для изображения.  
  
**5.** Можно запустить наше приложение. По нажатию на кнопку, картинка исчезнет, а потом медленно начнет прорисовываться обратно.  
  
**6.** Теперь рассмотрим подробно как же создается анимация в xml файле.  
Существует 4 вида анимации:

* **alpha** (прозрачность, видимость)
* **scale** (масштабирование)
* **rotate** (поворот)
* **translate** (перемещение)

Для создания анимации мы должны описать начальное и конечное состояние объекта, а система сама решит как перейти из одного состояния в другое. В нашем примере

<alpha

android:fromAlpha="0.0"

android:toAlpha="1.0"

android:duration="1000"/>

мы описываем анимацию alpha, то есть изменяем видимость объекта. Задаем начальное состояние*fromAlpha="0.0"* (полностью невидимое) и конечное *toAlpha="1.0"* (полностью видимое). Указываем продолжительность анимации *duration="1000"* (в миллисекундах). А все остальное, то есть как нужно изменять видимость объекта чтобы за секунду сделать его из невидимого в видимое, система делает сама. Рассчитывается это с помощью **интерполяции** - в вычислительной математике способ нахождения промежуточных значений величины по имеющемуся дискретному набору значений. Для каждой анимации можно задать интерполятор  
**-AccelerateDecelerateInterpolator** (@android:anim/accelerate\_decelerate\_interpolator) - скорость изменения в начале и конце низкая, а в середине ускоряется  
  
**-AccelerateInterpolator** (@android:anim/accelerate\_interpolator) - скорость изменения в начале низкая, а затем ускоряется  
  
**-AnticipateInterpolator** (@android:anim/anticipate\_interpolator) - изменения начинаются в обратную сторону, а затем резко двигаются вперед  
  
**-AnticipateOvershootInterpolator** (@android:anim/anticipate\_overshoot\_interpolator) - изменения начинаются в обратную сторону, затем резко двигаются вперед и пролетают выше конечного значения, а затем возвращаются до конечного значения  
  
**-BounceInterpolator** (@android:anim/bounce\_interpolator) - скорость изменения увеличивается в конце  
  
**-CycleInterpolator** (@android:anim/cycle\_interpolator) - повторение анимации указанное число раз. Скорость изменения следует синусоиде  
  
**-DecelerateInterpolator** (@android:anim/decelerate\_interpolator) - скорость изменения уменьшается в конце  
  
**-LinearInterpolator** (@android:anim/linear\_interpolator) - скорость изменения постоянна  
  
**-OvershootInterpolator** (@android:anim/overshoot\_interpolator) - изменения резко двигаются вперед и пролетают выше конечного значения, а затем возвращаются до конечного значения  
  
Задается интерполятор с помощью атрибута android:interpolator. Например  
*android:interpolator="@android:anim/cycle\_interpolator"*. По умолчанию используется LinearInterpolator.  
  
**7. Описание начальных и конечных состояний**  
1) **alpha (прозрачность, видимость)**  
*- android:fromAlpha* — начальное значение прозрачности. 0.0 — полностью прозрачное (невидимое), 1.0 — полностью непрозрачное (видимое)  
*- android:toAlpha* — конечное значение прозрачности  
  
2) **scale (масштабирование)**  
*- android:fromXScale* — начальное значение масштаба по оси X (где текущий размер соответствует значению 1.0)  
*- android:toXScale* — конечное значение масштаба по оси X  
*- android:fromYScale* - начальное значение масштаба по оси Y (где текущий размер соответствует значению 1.0)  
*- android:toYScale* - конечное значение масштаба по оси Y  
*- android:pivotX* — х координата точки, которая останется неизменна после масштабирования  
*- android:pivotY* - y координата точки, которая останется неизменна после масштабирования  
  
Возможные значение pivotX и pivotY:   
в пикселях относительно левого (или верхнего для координаты Y) края элемента (например «5»)  
в процентах относительно левого (верхнего) края (например «5%»)  
в процентах относительно левого (верхнего) края родительского элемента (например «5%p»)  
  
Например, если pivotX=0, pivotY=0 (что соответствует верхнему левому углу элемента), то масштабирование будет изменять размер элемента вниз и вправо. Если pivotX=50%, pivotY=50%, то точка находится по центру элемента и размер изменяется во все сторону, при этом центр будет оставаться а в одной точке.  
  
3) **rotate (поворот)**  
*- android:fromDegrees* — Начальное значение угла поворота (в градусах, возможно отрицательное значение)  
*- android:toDegrees* — конечное значение угла поворота   
*- android:pivotX* — x координаты центра поворота.   
*- android:pivotY* — y координата центра поворота.   
Возможные значения pivotX и pivotY как у анимации scale  
  
4) **translate (перемещение)**  
*- android:fromXDelta* — x координата начальной точки перемещения. Возможные значения:  
в пикселях относительно изначальной позиции (например «5»)  
в процентах относительно ширины элемента (например «5%»)  
в процентах относительно ширины родительского элемента (например «5%p»)  
*- android:toXDelta* - x координата конечной точки перемещения  
*- android:fromYDelta* - y координата начальной точки перемещения  
*- android:toYDelta* - y координата конечной точки перемещения  
  
**8. Дополнительные параметры**  
Также есть атрибуты общие для всех четырех типов анимации, наиболее полезные из них:  
*- android:duration* - длительность анимации (в миллисекундах)  
*- android:interpolator* — определяет интерполятор для анимации  
*- android:repeatCount* - кол-во дополнительных повторений анимации. Именно дополнительных, то есть один раз анимация выполнится по любому. Значением по умолчанию является «0» - это значит анимация выполнится только один раз. Значение «1» значит, что анимация выполнится два раза (один раз основной и один раз дополнительный). Значение «-1» или «infinite» - бесконечный повтор.   
*- android:repeatMode* — определяет поведение анимации, когда она дошла до конца, а параметр repeatCount не равен 0. Есть два значения «restart» - анимация начинается заново и «reverse» - анимация пойдет в обратном порядке.  
*- android:startOffset* — задержка перед началом анимации (в миллисекундах)  
  
**9. Объединение нескольких анимаций**  
К элементу можно применить одновременно несколько типов анимаций. Например если мы напишем:

<?**xml** version="**1.0**" encoding="utf-8"?>

<set xmlns:android="http:**//**schemas.android.com/apk/res/android">

<alpha

android:fromAlpha="0.0"

android:toAlpha="1.0"

android:duration="1000"

/>

<rotate

android:fromDegrees="0"

android:toDegrees="360"

android:pivotX="50%"

android:pivotY="50%"

android:duration="1000" />

</set>

Картинка за 1 секунду изменит прозрачность (с полностью прозрачной до непрозрачной) и при этом повернется на 360 градусов.   
  
Анимациям можно выставлять разную длительность, например поставим *duration=5000* у анимации rotate. Теперь картинка будет поворачиваться гораздо медленнее, а прозрачность меняется все-также за секунду.  
  
С помощью *startOffset*, можно сделать анимации последовательными. Добавим к rotate атрибут *startOffset="1000"*(то есть сделаем задержку равную длительности первой анимации). Теперь картинка вначале за 1 секунду станет видимой, а затем только повернется на 360 градусов.  
  
Несколько анимаций можно объединять в наборы тегом . Один такой тег будет в файле всегда и является корневым. Для набора можно задавать следующие атрибуты:  
- *duration* (длительность), *repeatMode* (режим повторения) — эти атрибуты будут применяться для каждой анимации в наборе  
- *interpolator* — определяет интерполятор анимации и *shareInterpolator* — будет ли этот интерполятор применятся для каждой анимации в наборе (возможные значения «true» и «false»)  
- *startOffset* (задержка) — задержка для всего набора анимаций.  
К сожалению, к набору нельзя применить атрибут *repeatCount*, то есть повторить несколько раз набор анимаций не получится.  
Наборы могут быть любой вложенности.  
  
**10. Создание анимации без xml**  
Анимацию можно создать и без использования xml, непосредственно в коде программы. Для этого используются классы наследники Animation:  
1) [AlphaAnimation](http://developer.android.com/reference/android/view/animation/AlphaAnimation.html" \t "_self) для создания анимации alpha. Конструктор класса имеет вид  
*AlphaAnimation(float fromAlpha, float toAlpha)* где fromAlpha и toAlpha соответственно начальное и конечное значение прозрачности (от 0.0 до 1.0)  
  
2) [RotateAnimation](http://developer.android.com/reference/android/view/animation/RotateAnimation.html" \t "_self) для создания анимации rotate. Имеет три конструктора:  
- *RotateAnimation (float fromDegrees, float toDegrees)*  
где fromDegrees и toDegrees — начальный и конечный угол поворота. Центр поворота находится в точке (0,0)  
- *RotateAnimation (float fromDegrees, float toDegrees, float pivotX, float pivotY)*  
pivotX и pivotY — координаты центра поворота в пикселях относительно левой и верхней границы элемента.  
- *RotateAnimation (float fromDegrees, float toDegrees, int pivotXType, float pivotXValue, int pivotYType, float pivotYValue)*  
pivotXType и pivotYType формат в котором указывается значение. Задается константой Animation.ABSOLUTE (абсолютное значение относительно левой или верхней границы в пикселях), Animation.RELATIVE\_TO\_SELF (значение в процентах относительно размеров элемента, причем значение 1.0 соответствует 100%) или Animation.RELATIVE\_TO\_PARENT (значение в процентах относительно размеров родительского элемента, причем значение 1.0 соответствует 100%).  
pivotXValue и pivotYValue — значение для координаты центра поворота (с учетом формата)  
  
3) [ScaleAnimation](http://developer.android.com/reference/android/view/animation/ScaleAnimation.html" \t "_self) для создания анимации scale. Также имеет три конструктора  
- *ScaleAnimation(float fromX, float toX, float fromY, float toY)*  
- *ScaleAnimation(float fromX, float toX, float fromY, float toY, float pivotX, float pivotY)*  
- *ScaleAnimation(float fromX, float toX, float fromY, float toY, int pivotXType, float pivotXValue, int pivotYType, float pivotYValue)*  
  
Параметры fromX, toX, fromY, toY — аналогичны атрибутам для создания анимации в xml. Параметры pivotX, pivotY, pivotXType, pivotXValue, pivotYType, pivotYValue — аналогичны параметра для анимации rotate.  
  
4) [TranslateAnimation](http://developer.android.com/reference/android/view/animation/TranslateAnimation.html" \t "_self) для создания анимации translate. Имеет два конструктора  
- *TranslateAnimation(float fromXDelta, float toXDelta, float fromYDelta, float toYDelta)*  
- *TranslateAnimation(int fromXType, float fromXValue, int toXType, float toXValue, int fromYType, float fromYValue, int toYType, float toYValue)*  
Значения параметров должны быть понятны  
  
5) и последний класс [AnimationSet](http://developer.android.com/reference/android/view/animation/AnimationSet.html" \t "_self) для объединения анимаций. Его конструктор  
*AnimationSet(boolean shareInterpolator)*  
shareInterpolator — определяет должны ли анимации входящие в набор использовать общий (заданный для набора) интерполятор или каждая анимация определяет свой. Значение «true» в первом случае и «false» во втором.  
  
У каждого класса есть методы для определения дополнительные параметров анимации, например:  
*setDuration(long durationMillis)* - установление длительности  
*setInterpolator(Context context, int resID)* - установление интерполятора  
*setRepeatCount(int repeatCount)* - установление кол-ва повторов  
*setRepeatMode(int repeatMode)* - установление режима повтора  
*setStartOffset(long startOffset)* - установление задержки начала анимации  
Все методы можно посмотреть на [странице документации](http://developer.android.com/reference/android/view/animation/Animation.html)  
  
**11.** Создадим в коде анимацию, которая по нажатию на кнопку будет поворачивать картинку на случайный угол (от 0 до 360) и увеличивать до случайного размера (не более чем в два раза). Я для этого добавила еще одну кнопку randomButton

randomButton.setOnClickListener(new OnClickListener() {

@Override

**public** **void** onClick(View v) {

**Random** random = **new** **Random**(); //1

RotateAnimation rotate = **new** RotateAnimation (0, (**float**)random.nextInt(360),

Animation.RELATIVE\_TO\_SELF, 0.5f, Animation.RELATIVE\_TO\_SELF, 0.5f); //2

rotate.setDuration(1000); //3

rotate.setRepeatMode(Animation.REVERSE); //4

rotate.setRepeatCount(1); //5

**long** duration = rotate.computeDurationHint(); //6

**float** size = random.nextFloat() + 1.0; //7

ScaleAnimation scale = **new** ScaleAnimation(1.0f, size, 1.0f, size,

Animation.RELATIVE\_TO\_SELF, 0.5f, Animation.RELATIVE\_TO\_SELF, 0.5f); //8

scale.setDuration(1000);

scale.setStartOffset(duration); //9

AnimationSet set = **new** AnimationSet (**false**); //10

set.addAnimation(rotate); //11

set.addAnimation(scale);

image.startAnimation(set); //12

}

});

1) Создаем объект Random, для генерации случайных чисел. Про Random подробнее можно прочитать в документации, сейчас нас интересуют методы int nextInt(int n) — генерирующий целое число в диапозоне от 0 до n. И метод float nextFloat() - генерирующий вещественное число от 0 до 1.  
2) Создаем анимацию вращения. Начальный угол = 0, конечный угол = случайному числу от 0 до 360. Animation.RELATIVE\_TO\_SELF означает, что точку центра поворота мы будем указывать в процентах относительно ширины элемента. Не забываем, что значение 1.0 соотвествует 100%, а значит 0.5f — это 50%. Значит точка центра поврота будет посередине картинки.  
3) Задаем длительность анимации 1000 миллисекунд (это 1 секунда)  
4) Определяем режим повторения как Animation.REVERSE, то есть при повторении анимация пойдем в обратном порядке.  
5) Задаем кол-во дополнительных повторений = 1. Значит всего анимация повторится два раза, один раз в прямом порядке и один в обратном.  
6) Метод long computeDurationHint() расчитывает сколько суммарно будет продолжаться анимация. Есть метод getDuration(), но он просто возвращает значение длительности, которое мы задали методом setDuration(). В нашем случае мы задали значение длительности 1000 и метод getDuration() вернет 1000 и не учтет, что анимация будет повторяться два раза, а значит в действительности будет продолжаться 2000 миллисекунд. Метод computeDurationHint() рассчитает длительность с учетом повторов и задержек.  
7) Расчитываем новый размер картинки. Значение 1.0 это текущий масштаб картинки, значит значение 2.0 — увеличение картинки в два раза. Мы генерируем число от 0.0 до 1.0 и приплюсовываем 1, значит получаем число от 1.0 до 2.0  
8) Создаем анимацию масштабирования от текущего размера картинки до случайно сгенерированного число от 1.0 до 2.0  
9) Задаем задержку равную суммарной длительности анимации вращения. Чтобы вторая анимация начиналась сразу после окончания первой  
10) Создаем набор анимаций.  
11) Добавляем две созданные анимации в набор  
12) Применяем набор анимаций к картинке  
  
**12.** Еще один интересный метод класса Animation  
**setAnimationListener (Animation.AnimationListener listener)** — устанавливает слушателя изменений состояний анимации. Интерфейс Animation.AnimationListener определяет следующие методы:  
*onAnimationStart (Animation animation)* — вызывается при старте анимации  
*onAnimationRestart (Animation animation)* — вызывается при повторе анимации  
*onAnimationEnd (Animation animation)* — вызывается по окончании анимации  
  
Например:

anim = AnimationUtils.loadAnimation(this, R.anim.anim);

anim.setAnimationListener(new AnimationListener () {

@Override

**public** **void** onAnimationEnd(Animation animation) {

Log.d("MY", "animation end");

}

@Override

**public** **void** onAnimationRepeat(Animation animation) {

Log.d("MY", "animation repeat");

}

@Override

**public** **void** onAnimationStart(Animation animation) {

Log.d("MY", "animation start");

}

});

Ничего полезного мы при изменении состояния анимации не делаем, просто пишем это в лог.  
  
На этом все. Основное я рассказала, остальное лучше изучать экспериментами :)   
  
Исходники можно скачать тут [AnimationaTest.zip](http://yadi.sk/d/OrTB_Rcb5AH5M)