# 鸿蒙开源第三方组件——加载动画库AVLoadingIndicatorView

## 前言

基于安卓平台的加载动画库AVLoadingIndicatorView（<https://github.com/81813780/AVLoadingIndicatorView>），实现了鸿蒙化迁移和重构，代码已经开源到（https://gitee.com/iscas-ohos/avloading-indicator-view\_ohos.git），欢迎各位下载使用并提出宝贵意见！

## 背景

服务器在加载数据的时候有时需要等待一段时间，加载动画可以缓解用户等待过程中的焦虑情绪，使等待过程变得更加友好、流畅。AVLoadingIndicatorView是一个开源的加载动画库，通过动画库的调用，可以实现各种各样的加载效果。

## 组件功能展示

### 1. 动画效果

在原Android版本中，本组件库中共有28个加载动画，鸿蒙版本的组件库成功实现其中21种动画效果。由于鸿蒙系统部分API功能缺失，目前有7个动画效果未成功迁移。

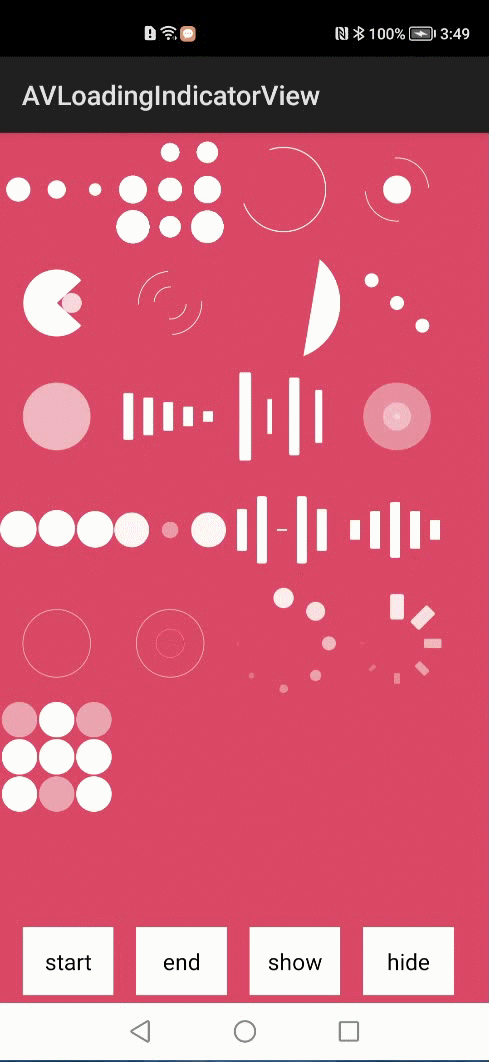


图1 AVLoadingIndicatorView效果示意图

图1所示为鸿蒙平台的AVLoadingIndicatorView的动画效果展示，可分为6行4列，动画效果的对应名称（从左至右）如下：

Row 1

BallPulseIndicator，BallGridPulseIndicator，BallClipRotateIndicator，BallClipRotatePulseIndicator

Row 2

PacmanIndicator，BallClipRotateMultipleIndicator，

SemiCircleSpinIndicator，BallRotateIndicator

Row 3

BallScaleIndicator，LineScaleIndicator，LineScalePartyIndicator，BallScaleMultipleIndicator

Row 4

BallPulseSyncIndicator，BallBeatIndicator，LineScalePulseOutIndicator，LineScalePulseOutRapidIndicator

Row 5

BallScaleRippleIndicator，BallScaleRippleMultipleIndicator，BallSpinFadeLoaderIndicator，LineSpinFadeLoaderIndicator

Row 6

BallGridBeatIndicator

### 2. 动画控制效果

AVLoadingIndicatorView组件支持对各加载动画的效果进行控制，控制类型分为4种：动画启动、动画停止、动画显示和动画隐藏。动画启动和动画停止相互关联，二者用于控制动画是否运行；动画显示和动画隐藏相互关联，二者用于控制动画是否出现，控制效果如图2所示。

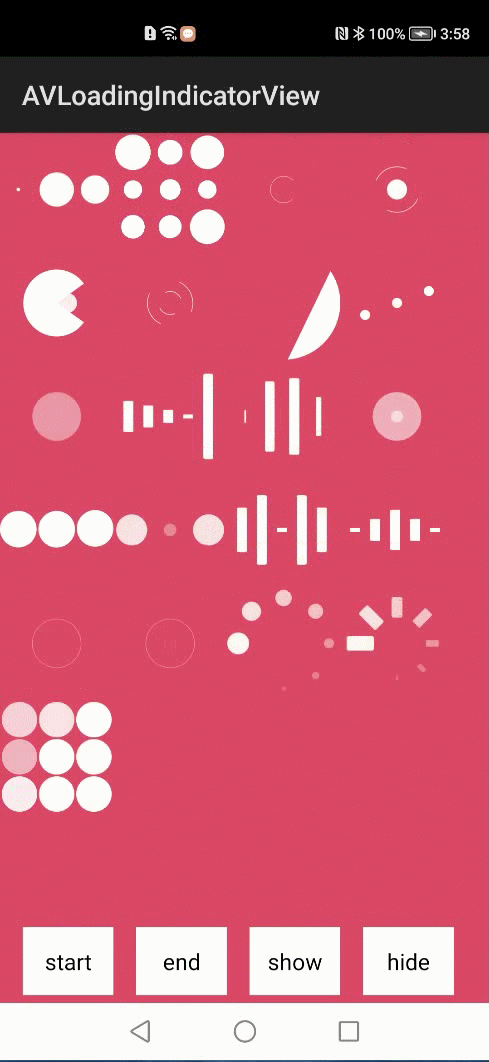


图2 四种效果控制示意图

## Sample解析

本组件库中的每个动画都有启动、停止、隐藏和显示四种控制效果。为了方便调试和演示，Sample中将所有动画的对象放入同一个list，通过四个不同的按钮，控制所有动画同时启动、停止、隐藏和显示，效果如图2所示。

下面介绍控制所有动画同时启动、停止、隐藏和显示的步骤：

1、导入AVLoadingIndicatorView类

**import** com.wang.avi.AVLoadingIndicatorView;

1. 动画添加到Layout

此处需要将所有动画添加到Layout中，并将各动画的对象放入同一list（即代码中的**animatorList**）。

//以BallPulseIndicator为例  
**myLayout**.addComponent(ballPulseIndicator);//BallPulseIndicator添加到Layout  
**animatorList**.add(ballPulseIndicator);//BallPulseIndicator对象放入list

//以BallGridPulseIndicator为例

**myLayout**.addComponent(ballGridPulseIndicator);  
**animatorList**.add(ballGridPulseIndicator);

1. 设置四个按钮，用以控制所有动画同时启动、停止、隐藏和显示。

以"启动"效果为例，startBtn按钮设置了Click监听，按下按钮时，会执行startAllAnimator ()方法，startAllAnimator ()方法中借助for循环，遍历**animatorList**中的每一个动画的对象，并调用每个对象的start（）方法，达到动画启动的效果。

//按钮监听

startBtn.setClickedListener(component-> startAllAnimator(**animatorList**));//启动  
endBtn.setClickedListener(component-> stopAllAnimator(**animatorList**));//停止

hideBtn.setClickedListener(component-> hideAllAnimator(**animatorList**));//隐藏  
showBtn.setClickedListener(component-> showAllAnimator(**animatorList**));//显示  
//start

private void startAllAnimator(ArrayList<AVLoadingIndicatorView> avLoadingIndicatorViews){  
 for (int i = 0; i < avLoadingIndicatorViews.size(); i++) {  
 avLoadingIndicatorViews.get(i).start();//启动  
 }  
}

//stop

private void stopAllAnimator(ArrayList<AVLoadingIndicatorView> avLoadingIndicatorViews){  
 for (int i = 0; i < avLoadingIndicatorViews.size(); i++) {  
 avLoadingIndicatorViews.get(i).stop();//停止  
 }  
}

//hide  
private void hideAllAnimator(ArrayList<AVLoadingIndicatorView> avLoadingIndicatorViews){  
 for (int i = 0; i < avLoadingIndicatorViews.size(); i++) {  
 avLoadingIndicatorViews.get(i).hide();//隐藏  
 }  
}  
//show  
private void showAllAnimator(ArrayList<AVLoadingIndicatorView> avLoadingIndicatorViews){  
 for (int i = 0; i < avLoadingIndicatorViews.size(); i++) {  
 avLoadingIndicatorViews.get(i).show();//显示  
 }  
}

## Library解析

### 1. 功能实现

每个动画效果的绘制都需要执行初始化设置、添加绘画任务、创建动画三个步骤，下面以BallPulseIndicator为例对这三个步骤进行详细介绍。

(1) 初始化设置

声明setPaint()，setBound()，draw()方法。

**public** BallPulseIndicator(Context context) {

**super**(context);  
 Component.DrawTask task = (component, canvas) -> {  
 setPaint();//设置画笔  
 setBound();//设置动画边界  
 draw(canvas,getPaint());//内容绘制  
 };  
 addDrawTask(task);  
}

(2) 动画绘制

用draw()方法在画布上绘制动画样式。由于BallPulseIndicator动画主体是三个圆，第二、第三个圆均是由前一个圆平移得到，因此要设置每两个圆之间的距离，圆的半径大小等属性。

**public void** draw(Canvas canvas, Paint paint) {  
 **float** circleSpacing=4; //设置圆之间距离**float** radius=(Math.*min*(getWidth(),getHeight())-circleSpacing\*2)/6; //设置圆的半径大小**float** x = getWidth()/ 2-(radius\*2+circleSpacing);//圆心的x轴坐标  
 **float** y=getHeight() / 2;//圆心的y轴坐标  
 **for** (**int** i = 0; i < 3; i++) {  
 canvas.save();  
 **float** translateX=x+(radius\*2)\*i+circleSpacing\*i;//平移后新圆心的x轴坐标  
 canvas.translate(translateX, y);  
 canvas.scale(**scaleFloats**[i], **scaleFloats**[i]);//缩放效果绘制  
 canvas.drawCircle(0, 0, radius, paint);//绘制圆  
 canvas.restore();  
 }  
}

(3) 动画参数设置

通过AnimatorValue对动画参数进行具体设置，包括动画的持续时间、重复次数、启动延迟时间、缩放和旋转等（BallPulseIndicator只涉及缩放而无旋转）。

**public** ArrayList<AnimatorValue> onCreateAnimators() {  
 ArrayList<AnimatorValue> animators=**new** ArrayList<>();  
 **int**[] delays=**new int**[]{120,240,360}; //设置三个圆的延迟时间  
 **for** (**int** i = 0; i < 3; i++) {  
 **final int** index=i;  
 AnimatorValue scaleAnim=**new** AnimatorValue(); //值动画  
 scaleAnim.setDuration(750); //动画持续时间  
 scaleAnim.setLoopedCount(-1); //动画无限次重复  
 scaleAnim.setDelay(delays[i]); //每个圆的延迟时间  
 addUpdateListener(scaleAnim, (animatorValue, v) -> {  
 **scaleFloats**[index] = v;//控制缩放  
 invalidate();//刷新界面  
 });  
 animators.add(scaleAnim);  
 }  
 **return** animators;  
}

### 2. 移植方法

(1) API直接替换

在安卓中使用ValueAnimator类设置加载动画的属性，移植之后这些功能主要基于鸿蒙的AnimatorValue类实现。这两个类中的方法名也不同，在进行鸿蒙化迁移时需要根据功能逐一替换。例如：鸿蒙中使用setLoopedCount()方法替换原setRepeatCount()方法来实现动画重复次数的设置。

(2) 函数重写

鸿蒙的AnimatorValue类相比较于安卓，缺少很多接口，若在实现部分复杂动画时，需要调用这些接口，只能采用函数重写的方法，这也是移植中的主要难点。如安卓中用ValueAnimator.ofFloat(1,0.5f,1)来设置动画的属性值1—0.5f—1的两次变化，实现动画的运行效果，而鸿蒙中缺少该接口，属性值只能在0—1之间单次变化，无法实现动画的完美效果，需要进行功能重写，下面给出此功能重写的代码。

public void onUpdate(AnimatorValue animatorValue, **float** v) {

if(v<=**0.5f**)

**scaleFloats**[**index**] =**1**-v;

else

**scaleFloats**[**index**] = v;

invalidate();

}

# **项目贡献人**

刘磊 郑森文 朱伟 陈美汝 王佳思