# CircleImageView组件功能介绍

## 功能介绍：

CircleImageView组件是一个加载圆形图片的组件，除去一般的Image设置外还支持边框颜色，边框宽度，背景颜色，是否允许图片压住边框四种自定义设置。

## 模拟器上运行效果：



# CircleImageView使用方法

## 新建工程，增加组件Har包依赖

在应用模块中添加HAR，只需要将circleimageview.har复制到entry\libs目录下即可（由于build.gradle中已经依赖的libs目录下的\*.har，因此不需要在做修改）。

## 修改主页面的布局文件

修改主页面的布局文件ability\_main.xml，将自定的CircleImageView添加到xml中，并对四个自定义属性赋值。修改后代码如下：



## 修改MainAbilitySlince的UI加载代码

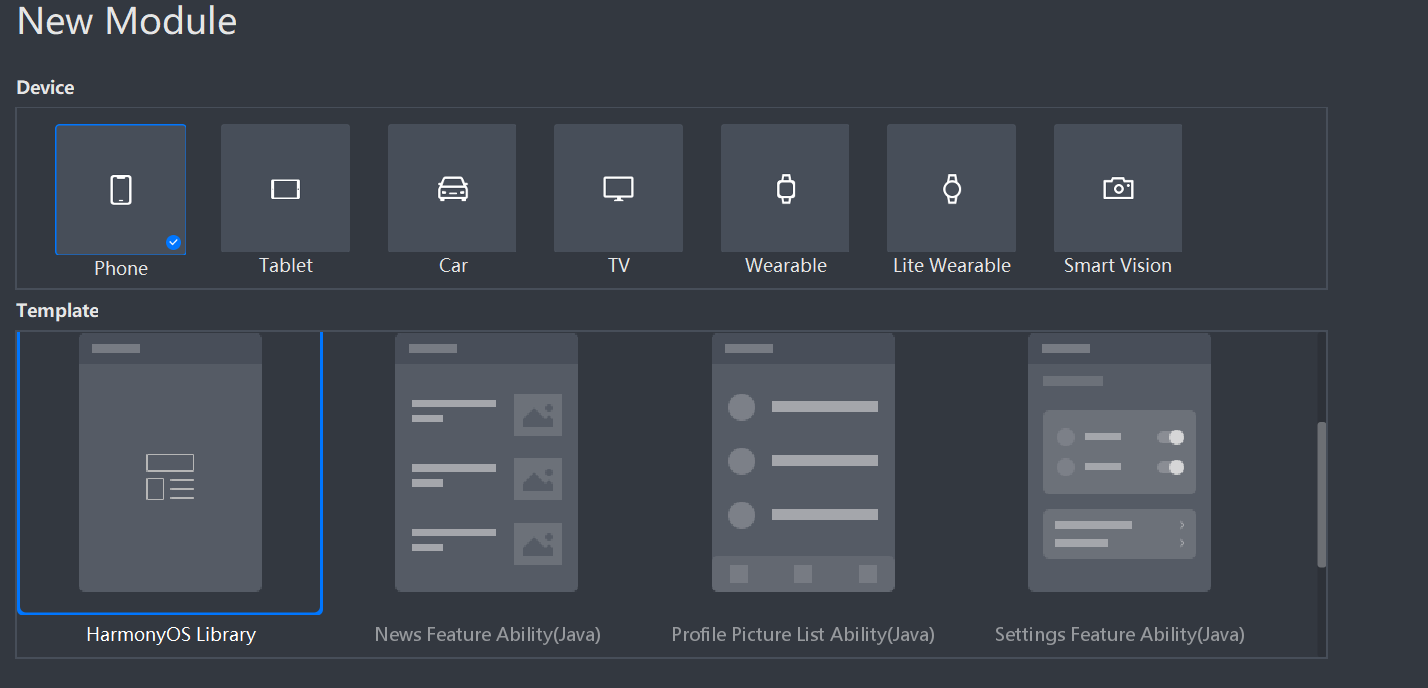
在MainAbilitySlince类的onStart函数中，增加如下代码。

**public void** onStart(Intent intent) {  
 **super**.onStart(intent);  
 **super**.setUIContent(ResourceTable.***Layout\_ability\_main***);  
 CircleImageView imageView = (CircleImageView) findComponentById(ResourceTable.***Id\_image***);  
 imageView.setImageAndDecodeBounds(ResourceTable.***Media\_hugh***);  
  
}

# CircleImageView开发实现

## 新建一个Module

新建一个Module，类型选择HarmonyOS Library，模块名为CircleImageView，如图



## 新建一个CircleImageView类

新建一个CircleImageView类，继承自Image类，继承Image.DrawTask, Image.TouchEventListener,Image.LayoutRefreshedListener方法，并重写他们的静态方法

**CircleImageView的主要流程：**

首先通过xml中设置image\_src方法设置或者代setImageAndDecodeBound方法通过处理获取位图PixelMap，用PixelMap构建像素图片持有者pixelMapHolder；

*//在设置图片时获取图片资源解码为无压缩的位图格式*@Override  
**public void** setPixelMap(PixelMap pixelMap) {  
 *// super.setPixelMap(pixelMap);* HiLog.*info*(*label*, **"Log\_setPixelMap"**);  
 **mPixelMap** = pixelMap;  
 initializePixelMap();  
}  
  
@Override  
**public void** setImageAndDecodeBounds(**int** resId) {  
 *// super.setImageAndDecodeBounds(resId);* **mPixelMap** = getPixelMap(resId);  
 initializePixelMap();  
 HiLog.*info*(*label*, **"Log\_setImageAndDecodeBounds"**);  
}

*/\*\*  
 \* 通过资源ID获取位图对象  
 \*\*/***private** PixelMap getPixelMap(**int** resId) {  
 InputStream drawableInputStream = **null**;  
 **try** {  
 drawableInputStream = getResourceManager().getResource(resId);  
 ImageSource.SourceOptions sourceOptions = **new** ImageSource.SourceOptions();  
 sourceOptions.**formatHint** = **"image/png"**;  
 ImageSource imageSource = ImageSource.*create*(drawableInputStream, **null**);  
 ImageSource.DecodingOptions decodingOptions = **new** ImageSource.DecodingOptions();  
 decodingOptions.**desiredSize** = **new** Size(0, 0);  
 decodingOptions.**desiredRegion** = **new** Rect(0, 0, 0, 0);  
 decodingOptions.**desiredPixelFormat** = PixelFormat.***ARGB\_8888***;  
 PixelMap pixelMap = imageSource.createPixelmap(decodingOptions);  
 **return** pixelMap;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 } **finally** {  
 **try** {  
 **if** (drawableInputStream != **null**) {  
 drawableInputStream.close();  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 **return null**;  
}

*/\*\*  
 \* 获取原有Image中的位图资源后重新设置PixelMapHolder  
 \*/***private void** initializePixelMap() {  
 HiLog.*info*(*label*, **"initializePixelMap"**);  
 **if** (**mPixelMap** != **null**) {  
 **pixelMapHolder** = **new** PixelMapHolder(**mPixelMap**);  
 **if** (**mInitialized**) {  
 updateShaderMatrix();  
 }  
 invalidate();*//重新检验该组件* } **else** {  
 HiLog.*info*(*label*, **"Log\_mPixelMapNULL"**);  
 **pixelMapHolder** = **null**;  
 }  
}

重写CircleImageView()的构造函数获取自定义参数，设置TouchEvent监听和LayoutRefreshed监听，以及调用init()函数进行画笔的初始化，以及调用updateDimensions()函数和invalidate()函数； LayoutRefreshed触发时也会调用updateDimensions()函数和invalidate()函数；

**public** CircleImageView(Context context, AttrSet attrSet, String styleName) {  
 **super**(context, attrSet, styleName);  
 *//获取xml中设置的自定义属性值* **mBorderWidth** = attrSet.getAttr(**"civ\_border\_width"**).get().getIntegerValue();  
 **mBorderColor** = attrSet.getAttr(**"civ\_border\_color"**).get().getColorValue();  
 **mBorderOverlay** = attrSet.getAttr(**"civ\_border\_overlay"**).get().getBoolValue();  
 **mCircleBackgroundColor** = attrSet.getAttr(**"civ\_circle\_background\_color"**).get().getColorValue();  
  
 init();  
 *//设置TouchEvent监听* setTouchEventListener(**this**::onTouchEvent);  
 *//设置Refreshed监听* setLayoutRefreshedListener(**this**::onRefreshed);  
}

*//布局变化时重新计算画布范围***private void** updateDimensions() {  
 HiLog.*info*(*label*, **"Log\_updateDimensions"**);  
 **mBorderRect**.modify(calculateBounds());  
 **mBorderRadius** = Math.*min*((**mBorderRect**.getHeight() - **mBorderWidth**) / 2.0f, (**mBorderRect**.getWidth() - **mBorderWidth**) / 2.0f);  
 **mDrawableRect**.modify(**mBorderRect**);  
 *//图片不能压住圆环并且边框宽度不为0则将图片缩小* **if** (!**mBorderOverlay** && **mBorderWidth** > 0) {  
 HiLog.*info*(*label*, **"Log\_shouldinSet"**);  
 **mDrawableRect**.shrink(**mBorderWidth** - 1.0f, **mBorderWidth** - 1.0f);  
 }  
 **mDrawableRadius** = Math.*min*(**mDrawableRect**.getHeight() / 2.0f, **mDrawableRect**.getWidth() / 2.0f);  
 updateShaderMatrix();  
}

进入updateShaderMatrix()函数，计算缩放比例和平移，设置PixelMapShader的Matrix参数等；

*/\*\*  
 \* 这个函数为获取pixelMapShader的Matrix参数，设置最小缩放比例，平移参数。  
 \* 作用：保证图片损失度最小和始终绘制图片正中央的那部分  
 \*/***private void** updateShaderMatrix() {  
 *// mPixelMap = getPixelMap();* **if** (**mPixelMap** == **null**) {  
 HiLog.*info*(*label*, **"Log\_mPixelMapNULL"**);  
 **return**;  
 }  
  
 **float** scale;  
 **float** dx = 0;  
 **float** dy = 0;  
  
 **mShaderMatrix**.setMatrix(**null**);  
  
 **int** pixelMapHeight = **mPixelMap**.getImageInfo().**size**.**height**;  
 **int** pixelMapWidth = **mPixelMap**.getImageInfo().**size**.**height**;  
  
 **if** (pixelMapWidth \* **mDrawableRect**.getHeight() > **mDrawableRect**.getWidth() \* pixelMapHeight) {  
 scale = **mDrawableRect**.getHeight() / (**float**) pixelMapHeight;  
 dx = (**mDrawableRect**.getWidth() - pixelMapWidth \* scale) \* 0.5f;  
 } **else** {  
 scale = **mDrawableRect**.getWidth() / (**float**) pixelMapWidth;  
 dy = (**mDrawableRect**.getHeight() - pixelMapHeight \* scale) \* 0.5f;  
 }  
  
 **mShaderMatrix**.setScale(scale, scale);  
 **mShaderMatrix**.postTranslate((**int**) (dx + 0.5f) + **mDrawableRect**.**left**, (**int**) (dy + 0.5f) + **mDrawableRect**.**top**);  
 HiLog.*info*(*label*, **"Log\_mPixelMapEND"**);  
 **mRebuildShader** = **true**;  
}

在invalidate()函数触发ondraw()函数完成最终的绘制。使用配置好的Paint先画出绘制内圆形来以后再画边界圆形和背景圆。

@Override  
**public void** invalidate() {  
 *//添加绘制任务* addDrawTask(**this**::onDraw);  
}  
  
@Override  
**public void** onDraw(Component component, Canvas canvas) {  
 *//绘制背景圆* **if** (**mCircleBackgroundColor** != Color.***TRANSPARENT***) {  
 canvas.drawCircle(**mDrawableRect**.getHorizontalCenter(), **mDrawableRect**.getVerticalCenter(), **mDrawableRadius**, **mCircleBackgroundPaint**);  
 }  
 *//绘制边框* **if** (**mBorderWidth** > 0) {  
 canvas.drawCircle(**mBorderRect**.getHorizontalCenter(), **mBorderRect**.getVerticalCenter(), **mBorderRadius**, **mBorderPaint**);  
 }  
 **if** (**pixelMapHolder** == **null**) {  
 **return**;  
 }  
 **synchronized** (**pixelMapHolder**) {  
 *// 绘制图片，使用之前计算的数据* **if** (**mRebuildShader**) {  
 **mRebuildShader** = **false**;  
 PixelMapShader bitmapShader = **new** PixelMapShader(**pixelMapHolder**, Shader.TileMode.***CLAMP\_TILEMODE***, Shader.TileMode.***CLAMP\_TILEMODE***);  
 bitmapShader.setShaderMatrix(**mShaderMatrix**);  
 **mPixelMapPaint**.setShader(bitmapShader, Paint.ShaderType.***PIXELMAP\_SHADER***);  
 }  
 HiLog.*info*(*label*, **"Log\_圆图绘制"**);  
 canvas.drawCircle(**mDrawableRect**.getHorizontalCenter(), **mDrawableRect**.getVerticalCenter(), **mDrawableRadius**, **mPixelMapPaint**);  
 **pixelMapHolder** = **null**;  
 }  
}

## 编译HAR包

在Bulid选项中选择BuildAPP(s)/Hap(s)选择debug或者realse

待构建任务完成后，可以在工程目录中的circleimageview> bulid > outputs > har目录中，获取生成的HAR包。