Lottie进阶和原理分析

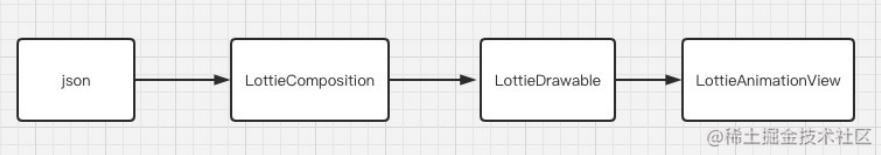
原创 蚂蚁二娘 于 2022-08-08 17:24:45 发布 886 收藏 1 版权
文章标签： android 动画

简介

Lottie是aribnb发布的开源库，它可以将AE制作的动画在Android、iOS和RN代码中渲染出来。

Lottie的功能及其强大，只需要设计师使用AE设计动画，用bodymovin导出，那么我们只需要简单的几行代码，就能实现非常复杂的动画效果。

LottieAnimationView继承自ImageView，通过当前时间绘制canvas显示到界面上。这里有两个关键类：LottieComposition 负责解析json描述文件，把json内容转成Java数据对象；LottieDrawable负责绘制，把LottieComposition转成的数据对象绘制成drawable显示到View上。顺序如下：



核心类：

- `LottieAnimationView` 继承自 ImageView ， 并且是加载 Lottie 动画的默认和最简单的方法。
- `LottieDrawable` 与 LottieAnimationView 有大部分相同的 API，但你可以在任何你想要的视图上使用它。
- `LottieComposition` 是动画的无状态model。只要你需要，此文件就可以安全地缓存，并且可以在drawable/view之间自由重用。
- `LottieCompositionFactory` 允许您从多个输入创建 LottieComposition。这就是 `setAnimation(...)` API 在后台使用 `LottieDrawable` 和 `LottieAnimationView` 使用的内容。工厂方法也与这些类共享相同的缓存。

参考文档

airbnb.io/lottie/#/an…

Lottie的使用方法

加载动画资源的方式：

- src/main/res/raw 中的 json 动画
- src/main/assets 中的 json 文件
- src/main/assets 中的 zip 文件
- src/main/assets中的dotLottie文件(*将Lottie的所有资源打包为一个.lottie文件，有兴趣可查看相关文档)
- json 或 zip 文件的 Url
- json 字符串
- json 或 zip 文件的 InputStream

xml中使用方法

（不再赘述）

xml文件中Lottie的各属性

属性	功能
lottie_fileName	设置播放动画的json文件名称
Lottie_rawRes	设置播放动画的json文件资源
Lottie_autoPlay	设置动画是否自动播放（默认为FALSE）
Lottie_loop	设置动画是否循环（默认为FALSE）
Lottie_repeatMode	设置动画的重复模式（默认为restart）
lottie_repeatCount	设置动画的重复次数（默认为-1）
Lottie_cacheStrategy	设置动画的缓存策略（默认为weak）

属性	功能
Lottie_colorFilter	设置动画的着色颜色（优先级最低）
Lottie_scale	设置动画的比例（默认为1f）
Lottie_progress	设置动画的播放进度
Lottie_imageAssetsFolder	设置动画依赖的图片资源文件地址

代码中使用Lottie

```
LottieAnimationView animationView = ...

animationView.setAnimation(R.raw.hello_world);
// or
animationView.setAnimation(R.raw.hello_world.json);

animationView.playAnimation();
```

缓存动画

默认情况下，所有Lottie动画都使用LRU缓存算法进行缓存，所有从raw或者assets文件夹加载出的动画都将默认创建缓存Key，其他API需要设置缓存key。如果需要对同一个动画并行触发多个动画请求，后续请求将加入现有任务，因此只会被解析一次。

全局配置

Lottie 有一些全局配置选项。默认情况下不需要，但它可用于：

- 从网络加载动画时，使用你自己的网络堆栈而不是 Lottie 的内置堆栈。
- 为从网络获取的动画提供您自己的缓存目录，而不是使用 Lottie 的默认目录 (`cacheDir/lottie_network_cache`)。
- 启用 systrace 进行调试。

要设置它，在应用程序初始化期间的某个地方，包括：

```
Lottie.initialize(
    LottieConfig.Builder()
      .setEnableSystraceMarkers(true)
      .setNetworkFetcher(...)
      .setNetworkCacheDir(...)
  )
```

注：systrace是Android自带的性能分析工具，详情可以查看文档

Android Systrace 系列文章

循环

Lottie可以通过setRepeatMode和setRepeatCount设置循环播放模式，或者通过在xml中设置 `lottie_loop="true"`

你同样可以循环动画中的某一段内容，通过调用 `setMinFrame` , `setMaxFrame` , or `setMinAndMaxFrame` ，包括帧、进度（从 0.0 到 1.0）或标记名称（在 After Effects 中指定）。

Lottie适配

Lottie 将 After Effects 中的所有 px 值转换为设备上的 dps，以便在设备上以相同大小呈现所有内容。这意味着，***Lottie本身已经自带了适配功能，*** *与其在 After Effects 中制作 1920x1080 的动画，不如在 After Effects 中制作 411x731px，大致对应于当今大多数手机的 dp 屏幕尺寸。

但是，如果您的动画尺寸不合适，您有两种选择：

0.**ImageView scaleType*** LottieAnimationView 是一个包装好的 `ImageView` ，它支持 `centerCrop` , `centerInside` ， `fitXY` 所以你可以像使用imageview一样使用此属性。
1.**Scaling Up/Down*** `LottieAnimationView` 和 `LottieDrawable` 两者都有一个 `setScale(float)` API，您可以使用它来手动放大或缩小动画。这很少有用，但在某些情况下可能有用。如果您的动画执行缓慢，请务必查看有关性能的文档。但是，请尝试结合 scaleType 缩小动画。这将减少 Lottie 每帧渲染的数量，特别是Lottie有大的mask或matters，这将特别有用。### 高级用法：动态修改属性

你可以在程序运行时动态更新Lottie属性，这可用于多种目的：

- 主题（白天、黑夜或任意主题）
- 响应成功或错误等事件
- 对动画的单个部分进行动画处理以响应事件

- 响应设计时未知的View大小或者其他属性

理解AE（After Effects）

要了解如何在 Lottie 中更改动画属性，首先应该了解动画属性是如何存储在 Lottie 中的。动画属性存储在模仿 After Effects 信息层次结构的数据树中。在 After Effects 中，`Composition`是一个集合`Layers`，每个集合都有自己的时间线。`Layer`对象具有字符串名称，它们的内容可以是图像、形状图层、填充、描边或任何可绘制的内容。After Effects 中的每个对象都有一个名称。Lottie可以使用这些对象和属性的名称通过`KeyPath`找到它们。

Lottie json文件的属性含义

- lottie的最外层结构：

```
{
  "v": "5.8.0",  //bodymovin的版本
  "fr": 60,      //帧率
  "ip": 0,       //起始关键帧
  "op": 102,     //结束关键帧
  "w": 1350,     //动画宽度
  "h": 800,      //动画高度
  "nm": "recommend_turn page_x0.75_original", //名称
  "ddd": 0,       //是否为3d
  "assets":[],   //资源信息
  "layers":[],   //图层信息
  "markers": []  //遮罩
}
注：时间=(op-ip)/fr
```

- assets

```
"assets": [         //资源信息
    {
        "id": "image_0",  //图片id
        "w": 129, //图片宽度
        "h": 884, //图片高度
        "u": "images/",   //图片路径
        "p": "recommend_bg_book_shadow.png",  //名称
        "e": 0
    },
}
```

- layers：动画是由一个一个的图层组合起来，并在图层上进行偏移、缩放等操作来实现动画的。图层的解析是lottie的主要功能模块。

```
 "layers": [//图层信息
  {
      "ddd": 0,         //是否为3d
      "ind": 1, //图层id 唯一性
      "ty": 4,            //图层类型
      "nm": "page back 4",//图层名称
      "refId": "comp_0", // 引用的资源，图片/预合成层
      "td": 1,
      "sr": 1,
      "ks": {...},// 变换。对应AE中的变换设置
      "ao": 0,
      ”layer“: [],          // 该图层包含的子图层
      “shaps”: [],          // 形状图层
      "ip": 12, //该图层起始关键帧
      "op": 1782,         //该图层结束关键帧
      "st": -18,
      "bm": 0
  }
```

- ks：对应AE中图层的变换属性，可以通过设置锚点、位置、旋转、缩放、透明度等来控制图层，并设置这些属性的变换曲线，来实现动画。

```
"ks": { // 变换。对应AE中的变换设置
    "o": { // 透明度
        "a": 0,
        "k": 100,
        "ix": 11
  },
    "r": { // 旋转
        "a": 0,
```

```
        "k": 0,
        "ix": 10
    },
    "p": { // 位置
        "a": 0,
        "k": [-167, 358.125, 0],
        "ix": 2
    },
    "a": { // 锚点
        "a": 0,
        "k": [667, 375, 0],
        "ix": 1
    },
    "s": { // 缩放
        "a": 0,
        "k": [100, 100, 100],
        "ix": 6
    }
}
```

- shape：对应AE中图层的内容中的形状设置的内容，其主要用于绘制图形

```
"shapes": [{
  "ty": "gr", // 类型。混合图层
  "it": [{ // 各图层json
      "ind": 0,
      "ty": "sh", // 类型，sh表示图形路径
      "ix": 1,
      "ks": {
          "a": 0,
          "k": {
              "i": [ // 内切线点集合
                [0, 0],
                [0, 0]
            ],
              "o": [ // 外切线点集合
                [0, 0],
                [0, 0]
            ],
              "v": [ // 顶点坐标集合
                [182, -321.75],
                [206.25, -321.75]
            ],
              "c": false // 贝塞尔路径闭合
        },
          "ix": 2
    },
      "nm": "路径 1",
      "mn": "ADBE Vector Shape - Group",
      "hd": false},{
    "ty": "st", // 类型。图形描边
    "c": { // 线的颜色
        "a": 0,
        "k": [0, 0, 0, 1],
        "ix": 3
  },
    "o": { // 线的不透明度
        "a": 0,
        "k": 100,
        "ix": 4
  },
    "w": { // 线的宽度
        "a": 0,
        "k": 3,
        "ix": 5
  },
    "lc": 2, // 线段的头尾样式
    "lj": 1, // 线段的连接样式
    "ml": 4, // 尖角限制
    "nm": "描边 1",
    "mn": "ADBE Vector Graphic - Stroke",
    "hd": false}]
}]
```

动态修改属性方法：

如果需要在运行时动态修改属性，需要以下三点：

0.KeyPath
1.LottieProperty

2.LottieValueCallback

KeyPath

KeyPath用于定位特定内容或将要更新的一组内容。KeyPath由字符串列表指定，这些字符串对应于原始动画中After Effectsd的内容层级结构。

KeyPaths 可以包含内容的特定名称或通配符：

- Wildcard(通配符)* 通配符匹配其在keypath中位置的任意单个内容名称
- Globstar(全局星标)* globstar匹配0个或多个层级。

KeyPath resolution

KeyPath能够存储对其解析的内容的内部引用。当您创建一个新的KeyPath对象时，它将被解析。LottieDrawable和LottieAnimationView有一个resolveKeyPath()方法，它接受一个KeyPath并返回一个由零个或多个已解析的KeyPath组成的列表，每个都在内部解析为一个内容片段。如果你不知道，这可以用来发现你的动画结构。为此，在开发环境中，解析新的KeyPath("")**并记录返回的列表。然而，你不应该单独使用**和ValueCallback，因为它会被应用到动画中的每一个内容片段。如果您解析了您的keypath，并希望随后添加一个值回调，请使用从该方法返回的keypath，因为它们将在内部解析，而不需要执行树遍历来再次查找内容。

LottieProperty

LottieProperty 是可以设置的属性的枚举。它们对应于 After Effects 中的动画值，可用属性在上面和文档中列出`LottieProperty`

以下属性可以运行时修改：

Transform	Layer	Fill	Stroke	
TRANSFORM_ANCHOR_POINT	TRANSFORM_ANCHOR_POINT	COLOR	COLOR	EL
TRANSFORM_POSITION	TRANSFORM_POSITION	OPACITY	OPACITY	
TRANSFORM_OPACITY	TRANSFORM_OPACITY	COLOR_FILTER	COLOR_FILTER	
TRANSFORM_SCALE	TRANSFORM_SCALE		STROKE_WIDTH	
TRANSFORM_ROTATION	TRANSFORM_ROTATION			
	TIME_REMAP			

ValueCallback

ValueCallback 是每次渲染动画时调用的内容。回调提供：

0.当前关键帧的起始帧。
1.当前关键帧的结束帧。
2.当前关键帧的起始值。
3.当前关键帧的结束值。
4.当前关键帧中从 0 到 1 的进度，没有任何时间插值。
5.当前关键帧的进度(存在插值器）。
6.整体动画进度从0到1。

ValueCallback类

- LottieValueCallback:可以在构造函数中设置静态值，也可以覆盖getValue()来设置每一帧的值。
- LottieRelativeTYPEValueCallback:可以在构造函数中设置一个静态值，也可以覆盖getOffset()来设置一个值，该值将被应用于每一帧上的实际动画值的偏移量。TYPE与LottieProperty参数的类型相同。
- LottieInterpolatedTYPEValue:提供一个开始值、结束值和可选的插值器，使值在整个动画中自动插入。TYPE与LottieProperty参数的类型相同。

动态修改属性的用法：

- **动态修改颜色**

```
KeyPath shirt = new KeyPath("Shirt", "Group 5", "Fill 1");
turnpagesLotv.addValueCallback(shirt, LottieProperty.COLOR, new LottieValueCallb
    @Nullable
    @Override
    public Integer getValue(LottieFrameInfo<Integer> frameInfo) {
             return frameInfo.getOverallProgress() > 0.5f ?
                       COLORS[index] :
                       COLORS[index++];
  }
```

```
});
```

- **修改弹跳高度**

```
private void setJumpHeight(){
       final PointF pointF = new PointF();
       mAnimationView.addValueCallback(new KeyPath("Body"), LottieProperty.TRAN
                                       new SimpleLottieValueCallback<PointF>()
           @Override
           public PointF getValue(LottieFrameInfo<PointF> frameInfo) {
               float startX = frameInfo.getStartValue().x;
               float startY = frameInfo.getStartValue().y;
               float endY = frameInfo.getEndValue().y;

               if (startY > endY) {
                   startY += mJmupArray[mIndex];
             } else if (endY > startY) {
                   endY += mJmupArray[mIndex];
             }
               pointF.set(startX, MiscUtils.lerp(startY, endY, frameInfo.getInt
               return pointF;
         }
     });
 }
```

- **事件绑定** （与手势事件绑定，本质上还是对position进行操作）

```
LottieRelativePointValueCallback largeValueCallback = new LottieRelativePointVa
       lottieAnimationView.addValueCallback(new KeyPath("First"),
               LottieProperty.TRANSFORM_POSITION, largeValueCallback);

       LottieRelativePointValueCallback mediumValueCallback = new LottieRelativ
       lottieAnimationView.addValueCallback(new KeyPath("Fourth"),
               LottieProperty.TRANSFORM_POSITION, mediumValueCallback);

       LottieRelativePointValueCallback smallValueCallback = new LottieRelative
       lottieAnimationView.addValueCallback(new KeyPath("Seventh"),
               LottieProperty.TRANSFORM_POSITION, smallValueCallback);

       ViewDragHelper viewDragHelper = ViewDragHelper.create(container, new Vie
           @Override
           public boolean tryCaptureView(@NonNull View child, int pointerId) {
               return child == targetView;
         }

           @Override
           public int clampViewPositionVertical(@NonNull View child, int top, i
               return top;
         }

           @Override
           public int clampViewPositionHorizontal(@NonNull View child, int left
               return left;
         }
           /**
            * 拖动的这个View的位置发生变化
            *
            * @param changedView当前拖动的这个View
            * @param left         距离左边的距离
            * @param top        距离右边的距离
            * @param dx           x轴的变化量
            * @param dy           y轴的变化量
            */
           @Override
           public void onViewPositionChanged(@NonNull View changedView, int lef
               totalDx += dx;
               totalDy += dy;
               //控制的是圆心然后触发重新绘制,就是位置的距离转换一下设置给新的圆心
               //这个触摸绑定交互可能不具有参考意义，因为动画没有特别复杂，直接canvas画
               smallValueCallback.setValue(getPoint(totalDx, totalDy, 1.2f));
               mediumValueCallback.setValue(getPoint(totalDx, totalDy, 1f));
               largeValueCallback.setValue(getPoint(totalDx, totalDy, 0.75f));
         }
     });
       container.setViewDragHelper(viewDragHelper);
```

注意：KeyPath构造函数中的字符串对应Lottie的json文件内不同层级的nm字段，通过nm字段，Lottie可以定位到需要动态修改属性的位置，不过当Lottie资源复杂时，比较难以找到对应字段。

- **更换图片资源**

```
//imageId--图片资源id
lottieView.updateBitmap(imageId, bitmap);
```

源码分析

LottieAnimationView继承自AppCompatImageView，Lottie动画能够实现的核心在于LottieDrawable。

以下为Lottie工作的简要流程：

- **LottieComposition：** After Effects/Bodymovin合成模型，这是创建动画的序列化模型。它被设计成无状态、可缓存和可共享的，这是json文件转换后的结果。
- **LottieDrawable：** 将LottieComposition封装为可以调用draw()方法的BaseLayer。
- **BaseLayer：** 当LottieAnimationView需要绘制时，将会逐层调用BaseLayer，从而将图像绘制出来。

Lottie第一步： json解析

通过LottieAnimationView的setAnimation()方法，可以看到

```
public void setAnimation(@RawRes final int rawRes) {
  this.animationResId = rawRes;
  animationName = null;
  setCompositionTask(fromRawRes(rawRes));
}

public void setAnimation(final String assetName) {
    this.animationName = assetName;
    animationResId = 0;
    setCompositionTask(fromAssets(assetName));
}
```

进入fromAssets方法：

```
private LottieTask<LottieComposition> fromAssets(final String assetName) {
  if (isInEditMode()) { //避免可视化编辑报错问题
    return new LottieTask<>(new Callable<LottieResult<LottieComposition>>() {
      @Override public LottieResult<LottieComposition> call() {
        return cacheComposition ?
            LottieCompositionFactory.fromAssetSync(getContext(), assetName) : Lo
    }
  }, true);} else {
    //cacheComposition记录是否已缓存
    //最终拿到json文件的LottieComposition数据模型
    return cacheComposition ?
        LottieCompositionFactory.fromAsset(getContext(), assetName) : LottieComp
}
```

然后对字节流内容进行解析

```
LottieComposition composition = LottieCompositionMoshiParser.parse(reader);
    if (cacheKey != null) {
      LottieCompositionCache.getInstance().put(cacheKey, composition);
}
```

解析json字段

```
private static final JsonReader.Options NAMES = JsonReader.Options.of(
    "w", // 0
    "h", // 1
    "ip", // 2
    "op", // 3
    "fr", // 4
    "v", // 5
    "layers", // 6
    "assets", // 7
    "fonts", // 8
    "chars", // 9
    "markers" // 10
);
```

```
public static LottieComposition parse(JsonReader reader) throws IOException {
  float scale = Utils.dpScale();
  float startFrame = 0f;
  float endFrame = 0f;
```

```
  float frameRate = 0f;
  final LongSparseArray<Layer> layerMap = new LongSparseArray<>();
  final List<Layer> layers = new ArrayList<>();
  int width = 0;
  int height = 0;
  Map<String, List<Layer>> precomps = new HashMap<>();
  Map<String, LottieImageAsset> images = new HashMap<>();
  Map<String, Font> fonts = new HashMap<>();
  List<Marker> markers = new ArrayList<>();
  SparseArrayCompat<FontCharacter> characters = new SparseArrayCompat<>();
  ......
}
```

Lottie第二步： LottieAnimationView将解析后生成的LottieComposition对象传递给LottieDrawer

```
/**
 * 设置一个composition.
 * 如果这个视图使用R.attr.lottie_cacheComposition填充xml，则可以设置默认缓存策略。
 */
public void setComposition(@NonNull LottieComposition composition) {
  if (L.DBG) {
    Log.v(TAG, "Set Composition \n" + composition);}
  lottieDrawable.setCallback(this);

  this.composition = composition;
  ignoreUnschedule = true;
  //将解析后的LottieComposition传递给LottieDrawable
  boolean isNewComposition = lottieDrawable.setComposition(composition);
  ignoreUnschedule = false;
  enableOrDisableHardwareLayer();
  if (getDrawable() == lottieDrawable && !isNewComposition) {
    // We can avoid re-setting the drawable, and invalidating the view, since th
    // hasn't changed.
    //我们可以避免重新设置drawable，并使视图无效，因为合成并没有改变。
    return;} else if (!isNewComposition) {
    // The current drawable isn't lottieDrawable but the drawable already has th
    // 当前的drawable不是lottieDrawable，但drawable已经有正确的组成。
    setLottieDrawable();}

  // This is needed to makes sure that the animation is properly played/paused f
  // 需要确保动画在当前可见状态是正确播放/暂停。
  // It is possible that the drawable had a lazy composition task to play the an
  // became invisible. Comment this out and run the espresso tests to see a fail
  onVisibilityChanged(this, getVisibility());

  requestLayout();

  for (LottieOnCompositionLoadedListener lottieOnCompositionLoadedListener : lot
    lottieOnCompositionLoadedListener.onCompositionLoaded(composition);}
}
```

LottieDrawable将LottieComposition对象构造为CompositionLayer

```
public boolean setComposition(LottieComposition composition) {
  if (this.composition == composition) {
    return false;}

  isDirty = false;
  clearComposition();
  this.composition = composition;
  buildCompositionLayer();...
```

```
 private void buildCompositionLayer() {
    compositionLayer = new CompositionLayer(
        this, LayerParser.parse(composition), composition.getLayers(), compositi
    if (outlineMasksAndMattes) {
      compositionLayer.setOutlineMasksAndMattes(true);
  }}
```

CompositionLayer继承自Baselayer，并且在构造时会遍历所有layer图层，转换为BaseLayer对象。

```
public CompositionLayer(LottieDrawable lottieDrawable, Layer layerModel, List<La
    LottieComposition composition) {
  super(lottieDrawable, layerModel);...
  LongSparseArray<BaseLayer> layerMap =
      new LongSparseArray<>(composition.getLayers().size());
  BaseLayer mattedLayer = null;
  for (int i = layerModels.size() - 1; i >= 0; i--) {
```

```
    Layer lm = layerModels.get(i);
    BaseLayer layer = BaseLayer.forModel(this, lm, lottieDrawable, composition);
   ...}
}
```

这里通过BaseLayer的forModel方法，将BaseLayer的各个子类型抽象出来

```
@Nullable
static BaseLayer forModel(
    CompositionLayer compositionLayer, Layer layerModel, LottieDrawable drawable
  switch (layerModel.getLayerType()) {
    case SHAPE:
      return new ShapeLayer(drawable, layerModel, compositionLayer);
    case PRE_COMP:
      return new CompositionLayer(drawable, layerModel,
          composition.getPrecomps(layerModel.getRefId()), composition);
    case SOLID:
      return new SolidLayer(drawable, layerModel);
    case IMAGE:
      return new ImageLayer(drawable, layerModel);
    case NULL:
      return new NullLayer(drawable, layerModel);
    case TEXT:
      return new TextLayer(drawable, layerModel);
    case UNKNOWN:
    default:
      // Do nothing
      Logger.warning("Unknown layer type " + layerModel.getLayerType());
      return null;}
}
```

以下是Lottie的不同layer类型

到这里，LottieDrawable就通过CompositionLayer将各个类型的layer实例化，然后在LottieDrawable的draw()方法中完成所有图层的绘制

```
@RestrictTo(RestrictTo.Scope.LIBRARY_GROUP)
public void draw(Canvas canvas, Matrix matrix) {
  CompositionLayer compositionLayer = this.compositionLayer;
  if (compositionLayer == null) {
    return;}
  compositionLayer.draw(canvas, matrix, alpha);
}
```

Lottie第三步： 播放Lottie动画

通过LottieAnimationView的playAnimation方法可以看到，内部会调用LottieDrawable的playAnimation方法，然后会触发LottieValueAnimator的playAnimation方法。LottieValueAnimator实际也是一个ValueAnimator，所以本质上Lottie也是属性动画驱动的。

具体在LottieDrawable中可以看到，LottieValueAnimator调用updateListener后，会刷新CompositionLayer的progress。

```
private final ValueAnimator.AnimatorUpdateListener progressUpdateListener = new
  @Override
  public void onAnimationUpdate(ValueAnimator animation) {
    if (compositionLayer != null) {
      compositionLayer.setProgress(animator.getAnimatedValueAbsolute());
  }}
};
```

进入setProgress可以看到，CompositionLayer会遍历所有layer图层，并逐个调用其setProgress方法。

```
@Override public void setProgress(@FloatRange(from = 0f, to = 1f) float progress
  super.setProgress(progress);
  if (timeRemapping != null) {
```

```
    // The duration has 0.01 frame offset to show end of animation properly.
    // https://github.com/airbnb/lottie-android/pull/766
    // Ignore this offset for calculating time-remapping because time-remapping
    float durationFrames = lottieDrawable.getComposition().getDurationFrames() +
    float compositionDelayFrames = layerModel.getComposition().getStartFrame();
    float remappedFrames = timeRemapping.getValue() * layerModel.getComposition(
    progress = remappedFrames / durationFrames;}
  if (timeRemapping == null) {
    progress -= layerModel.getStartProgress();}
  //Time stretch needs to be divided if is not "__container"
  if (layerModel.getTimeStretch() != 0 && !"__container".equals(layerModel.getNa
    progress /= layerModel.getTimeStretch();}
  for (int i = layers.size() - 1; i >= 0; i--) {
    layers.get(i).setProgress(progress);}
}
```

进入BaseLayer的setProgress方法会发现，会调用所有BaseKeyframeAnimation的setProgress方法，并会在BaseLayer中回调调用invalidateSelf()方法。

```
private void invalidateSelf() {
  lottieDrawable.invalidateSelf();
}
```

回调invalidateSelf()方法后，LottieDrawable会回调draw方法

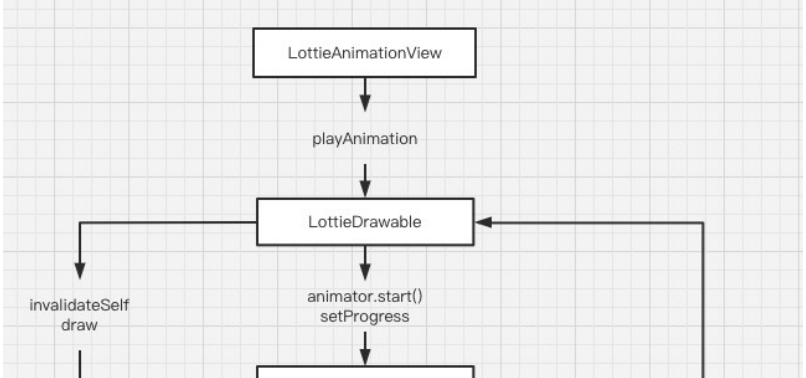
```
@RestrictTo(RestrictTo.Scope.LIBRARY_GROUP)
public void draw(Canvas canvas, Matrix matrix) {
  CompositionLayer compositionLayer = this.compositionLayer;
  if (compositionLayer == null) {
    return;}
  compositionLayer.draw(canvas, matrix, alpha);
}
```

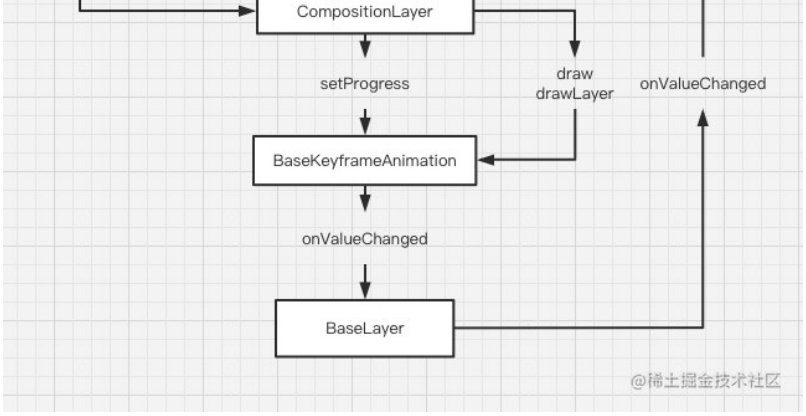
进入CompositionLayer的draw方法

```
@Override
public void draw(Canvas canvas, Matrix parentMatrix, int parentAlpha) {
  L.beginSection(drawTraceName);
  if (!visible || layerModel.isHidden()) {
    L.endSection(drawTraceName);
    return;}
  buildParentLayerListIfNeeded();
  L.beginSection("Layer#parentMatrix");
  matrix.reset();
  matrix.set(parentMatrix);
  for (int i = parentLayers.size() - 1; i >= 0; i--) {
    matrix.preConcat(parentLayers.get(i).transform.getMatrix());}
  L.endSection("Layer#parentMatrix");
  int opacity = transform.getOpacity() == null ? 100 : transform.getOpacity().ge
  int alpha = (int)
    ((parentAlpha / 255f * (float) opacity / 100f) * 255);
  if (!hasMatteOnThisLayer() && !hasMasksOnThisLayer()) {
    matrix.preConcat(transform.getMatrix());
    L.beginSection("Layer#drawLayer");
    drawLayer(canvas, matrix, alpha);
    L.endSection("Layer#drawLayer");
    recordRenderTime(L.endSection(drawTraceName));
    return;}
  ……
```

实际这样构成了一个循环，随着animator动画的进行，LottieDrawable会不断的绘制，这样Lottie动画就跑起来了，流程图如下：

流程图





Lottie性能优化

开发过程中经常会出现Lottie跳帧的问题，那么首先要明白，Lottie为何会跳帧?

进入LottieValueAnimator的playAnimation方法，可以看到

```
@MainThread
public void playAnimation() {
  running = true;
  notifyStart(isReversed());
  setFrame((int) (isReversed() ? getMaxFrame() : getMinFrame()));
  //lastFrameTimeNs这个时间戳代表上一帧动画的时间，第一帧为0
  lastFrameTimeNs = 0;
  repeatCount = 0;
  //开启动画之后，post了一个frameCallback
  postFrameCallback();
}
```

```
protected void postFrameCallback() {
  if (isRunning()) {
    removeFrameCallback(false);
    //每次界面绘制完一帧，都会回调一次这个接口，主流帧率监测的方案都是通过这个接口
    Choreographer.getInstance().postFrameCallback(this);}
}
```

继续顺藤摸瓜，找到FrameCallback的实现doFrame方法:

```
@Override public void doFrame(long frameTimeNanos) {
  postFrameCallback(); //重新回调
  if (composition == null || !isRunning()) {
    return;}

  L.beginSection("LottieValueAnimator#doFrame");
  //这里会拿到lastFrameTimeNs，计算两次进入回调后的时间差
  long timeSinceFrame = lastFrameTimeNs == 0 ? 0 : frameTimeNanos - lastFrameTim
  //动过这个时间差，计算下一帧的播放进度
  float frameDuration = getFrameDurationNs();
  float dFrames = timeSinceFrame / frameDuration;

  //这里便是跳帧发生的位置，frame代表帧数，如果dFrames这个时间差越大，那么frame的值也就越
  frame += isReversed() ? -dFrames : dFrames;
  boolean ended = !MiscUtils.contains(frame, getMinFrame(), getMaxFrame());
  frame = MiscUtils.clamp(frame, getMinFrame(), getMaxFrame());
  //这里重新标记上一帧的时间
  lastFrameTimeNs = frameTimeNanos;

  notifyUpdate();
  if (ended) {
    if (getRepeatCount() != INFINITE && repeatCount >= getRepeatCount()) {
      frame = speed < 0 ? getMinFrame() : getMaxFrame();
      removeFrameCallback();
      notifyEnd(isReversed());
  } else {
      notifyRepeat();
      repeatCount++;
      if (getRepeatMode() == REVERSE) {
        speedReversedForRepeatMode = !speedReversedForRepeatMode;
        reverseAnimationSpeed();
    } else {
        frame = isReversed() ? getMaxFrame() : getMinFrame();
    }
      lastFrameTimeNs = frameTimeNanos;
  }}
```

总结

理解了Lottie跳帧的机制，那么如何进行优化呢？

0.往往Lottie跳帧是主线程进行了耗时操作，那么最有方案便是优化此耗时操作，放到子线程等。
1.看Lottie的json结构，如果没有用到遮罩mask（掩膜）或者matte（前景蒙版）标签，那正常来讲性能开销没啥问题，这两个标签会创建bitmap，大幅拉高内存，特别是在recyclerview中。
2.**导出的矢量图层使用1x一倍图，这一点十分重要，Lottie会自动适配屏幕密度**
3.尽量保持图层简洁，预合成嵌套越少越好
4.开启硬件加速，lotv.setRenderMode(RenderMode.HARDWARE)，但是注意开启硬件加速后不支持抗锯齿、笔画上限（API 18前）和其他一些功能。