文章首先介绍Lottie的基本使用，然后分析把json文件映射到动画的实现思路，最后分析Lottie的源码实现，这里分析的是Lottie-Android。

基本用法

与使用相关的只有三个类文件：`LottieAnimationView、LottieComposition、LottieDrawable`，所以Lottie使用起来特别简单（需要注意Lottie支持API16及以上）。
最简单的使用方式是在xml中增加LottieAnimationView：

```
<com.airbnb.lottie.LottieAnimationView
    android:id="@+id/animation_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    app:lottie_fileName="Logo/LogoSmall.json"
    app:lottie_loop="true" />
```

"Logo/LogoSmall.json"是需要加载的动画数据路径，根目录是assets目录。

也可以通过代码设置动画数据json路径：

```
LottieAnimationView animationView =
        (LottieAnimationView) getView().findViewById(R.id.animation_view);
animationView.setAnimation("hello-world.json");
animationView.loop(true);
```

然后在代码中控制动画播放或者添加监听事件：

```
animationView.addAnimatorUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
    @Override
    public void onAnimationUpdate(ValueAnimator animation) {
        //do something
    }
});
animationView.playAnimation();

if (animationView.isAnimating()) {
    // Do something.
}

animationView.setProgress(0.5f);

// Custom animation speed or duration.
ValueAnimator animator = ValueAnimator.ofFloat(0f, 1f)
        .setDuration(500);

animator.addUpdateListener(animation -> {
    animationView.setProgress(animation.getAnimatedValue());
});
animator.start();
animationView.cancelAnimation();
```

Lottie提供了LottieDrawable可以使用：

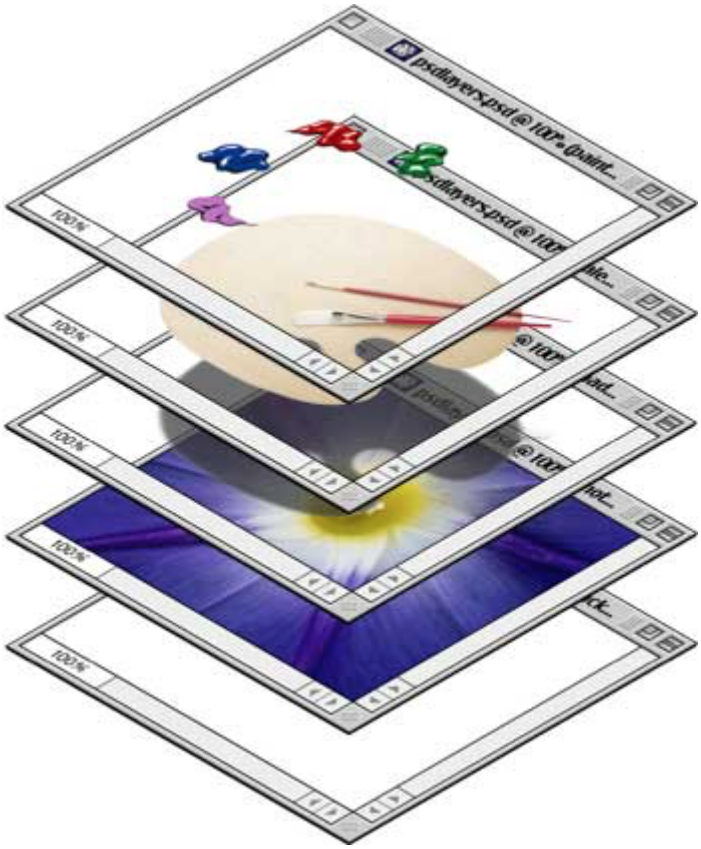
```
final LottieDrawable drawable = new LottieDrawable();
LottieComposition.fromAssetFileName(this, "hello-world.json",
        new LottieComposition.OnCompositionLoadedListener() {
            @Override
            public void onCompositionLoaded(LottieComposition composition) {
                drawable.setComposition(composition);
            }
        });
```

可以看到Lottie使用起来非常简单，我们之后就从以上用到的`LottieAnimationView`、`LottieComposition`、`LottieDrawable`入手来分析下Lottie动画的实现原理。

思路分析

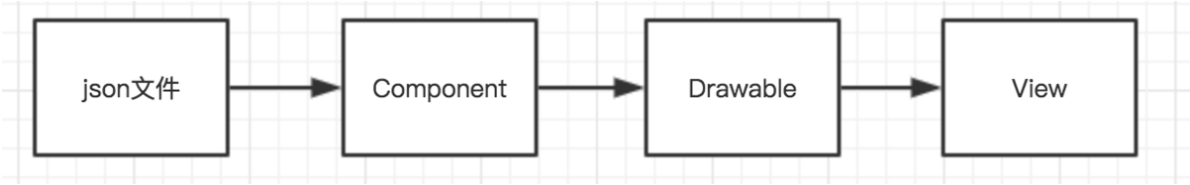
我们先从底层思考下如何在屏幕上绘制动画，最简单的方式是把动画分为多张图片，然后通过周期替换屏幕上绘制的图片来形成动画，这种暴力的方式非常简单，但缺点明显，很耗内存，动画播放中前后两张替换的图片在很多元素并没有变化，重复的内容浪费了空间。

为了提高空间利用率，可以把图片中的元素进行拆分，使用过photoshop的同学知道，其实在处理一张图片时，可以把一张复杂的图片使用多个图层来表示，每个图层上展示一部分内容，图层中的内容也可以拆分为多个元素。拆分元素之后，根据动画需求，可以单独对图层，甚至图层中的元素设置平移、旋转、收缩等动画。



Lottie使用json文件来作为动画数据源，json文件是通过Bodymovin 插件导出的，查看sample中给出的json文件，其实就是把图片中的元素进行来拆分，并且描述每个元素的动画执行路径和执行时间。Lottie的功能就是读取这些数据，然后绘制到屏幕上。

现在思考如果我们拿到一份json格式动画如何展示到屏幕上。首先要解析json，建立数据到对象的映射，然后根据数据对象创建合适的Drawable绘制到View上，动画的实现可以通过操作读取到的元素完成。



源码分析

1. json文件到对象的映射

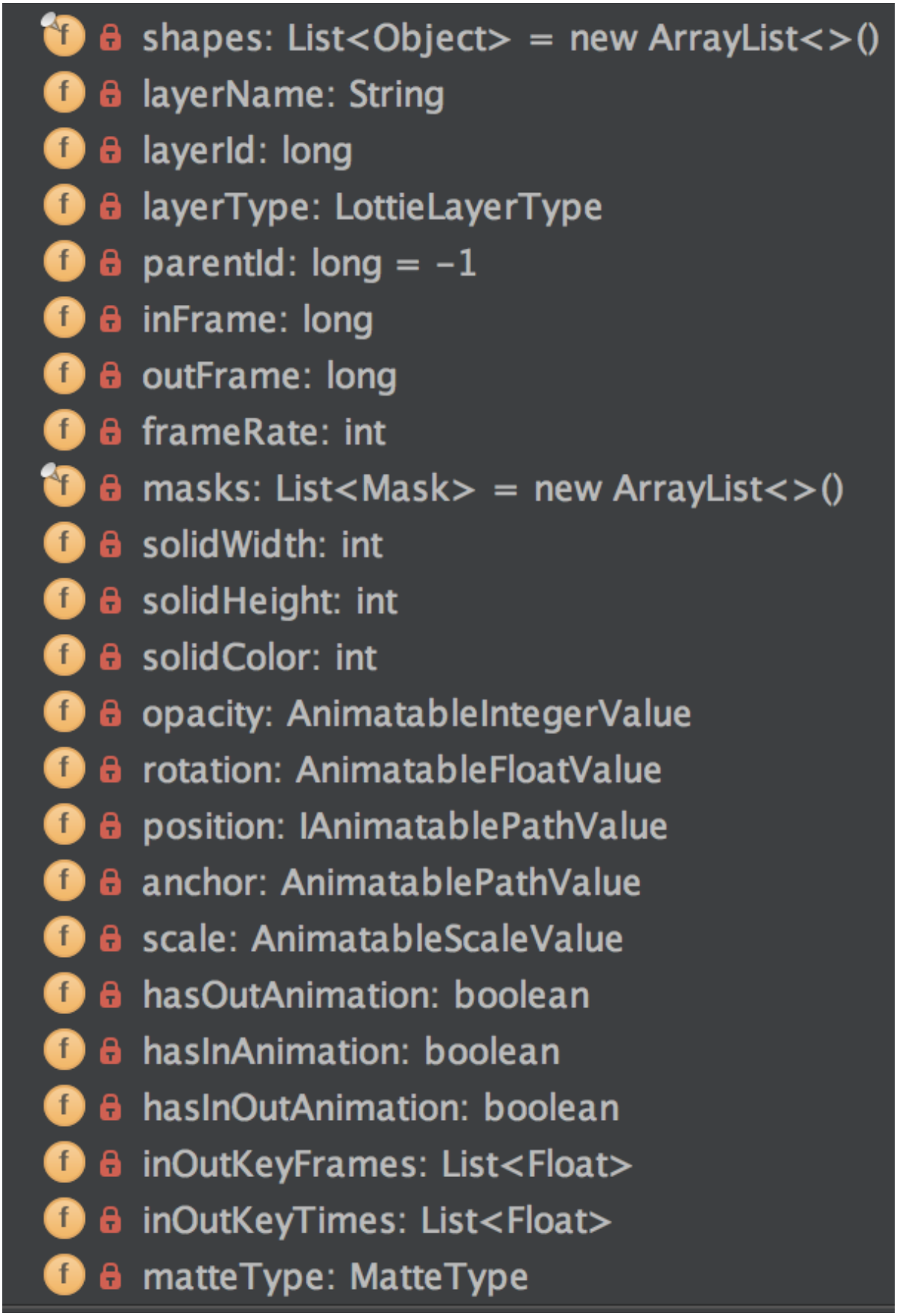
Lottie使用`LottieComposition`来作为After Effects的数据对象，即把json文件映射到`LottieComposition`，`LottieComposition`中提供了解析json的静态方法：

```
fromAssetFileName(Context, String, OnCompositionLoadedListener): Cancellable
fromInputStream(Context, InputStream, OnCompositionLoadedListener): Cancellable
fromFileSync(Context, String): LottieComposition
fromJson(Resources, JSONObject, OnCompositionLoadedListener): Cancellable
fromInputStream(Resources, InputStream): LottieComposition
fromJsonSync(Resources, JSONObject): LottieComposition
```

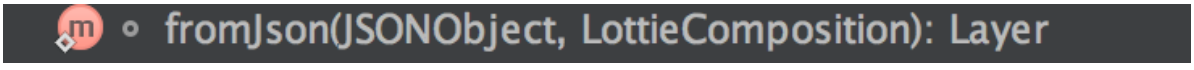
我们看下`LottieComposition`都有哪些成员变量，这些成员变量描述了After Effects中的动画。

```
layerMap: LongSparseArray<Layer> = new LongSparseArray<>()
layers: List<Layer> = new ArrayList<>()
bounds: Rect
startFrame: long
endFrame: long
frameRate: int
duration: long
hasMasks: boolean
hasMattes: boolean
scale: float
```

可以看到startFrame、endFrame、duration、scale等都是动画中常见的。我们看下`List<Layer>`，看名字就是映射拆分后的图层数据：



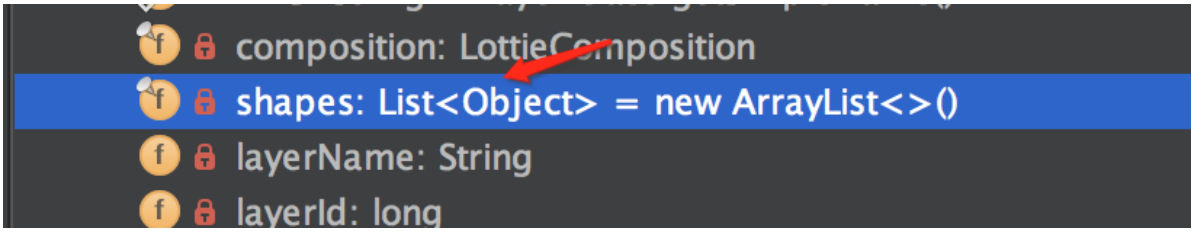
`Layer` 中完成layer的json数据解析：



2. 数据对象到Drawable的映射

`AnimatableLayer` 继承自 `Drawable`，我们看下它的子类：

其中`LayerView`对应着`Layer`数据，`Layer`中有



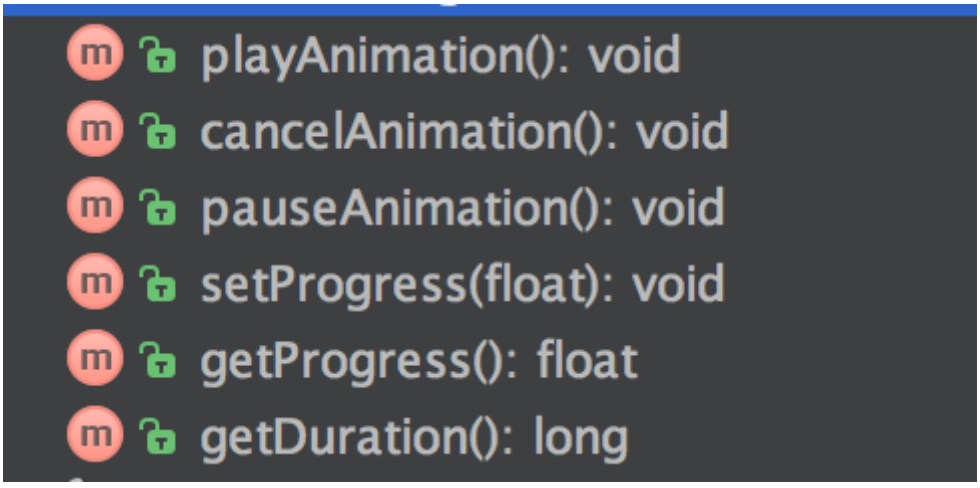
对应的`LayerView`中有

可以简单地理解为ViewGroup中可以包含ViewGroup或者View，但其实整个Lottie实现的动画都是绘制在一个View `LottieAnimationView`上。

`AnimatableLayer` 的其它子类如 `ShapeLayer`，`RectLayouer`等作为 `LayerView` 中 `List<AnimatableLayer>` 的元素。

3. 绘制

`LottieAnimationView` 继承自 `AppCompatImageView`，封装了一些动画的操作，如：



具体的绘制时委托为 `LottieDrawable` 完成的，我们看下 `LottieDrawable` 中的 `draw()` 方法：

```
@Override public void draw(@NonNull Canvas canvas) {
  if (composition == null) {
    return;
  }
  Rect bounds = getBounds();
  Rect compBounds = composition.getBounds();
  int saveCount = canvas.save();
  if (!bounds.equals(compBounds)) {
    float scaleX = bounds.width() / (float) compBounds.width();
    float scaleY = bounds.height() / (float) compBounds.height();
    canvas.scale(scaleX, scaleY);
  }
  super.draw(canvas);
  canvas.clipRect(getBounds());
  canvas.restoreToCount(saveCount);

}
```

`LottieDrawable` 继承自 `AnimatableLayer`，其 `draw()` 方法如下：

```
@Override
public void draw(@NonNull Canvas canvas) {
  int saveCount = canvas.save();
  applyTransformForLayer(canvas, this);

  int backgroundAlpha = Color.alpha(backgroundColor);
  if (backgroundAlpha != 0) {
    int alpha = backgroundAlpha;
    if (this.alpha != null) {
      alpha = alpha * this.alpha.getValue() / 255;
    }
    solidBackgroundPaint.setAlpha(alpha);
    if (alpha > 0) {
      canvas.drawRect(getBounds(), solidBackgroundPaint);
    }
  }
  for (int i = 0; i < layers.size(); i++) {
    layers.get(i).draw(canvas);
  }
  canvas.restoreToCount(saveCount);
}
```

可以看到先绘制了本层的内容，然后开始绘制包含的 `layers` 的内容：

```
for (int i = 0; i < layers.size(); i++) {
    layers.get(i).draw(canvas);
}
```

这个过程于界面中ViewGroup嵌套绘制类似。

实现分析

上面我们根据动画绘制的思路分析了下Lottie实现机制，下面从正面来捋一下程序的执行过程：

1. 创建`LottieAnimationView lottieAnimationView`
2. 创建`LottieDrawable lottieDrawable`
3. 使用`LottieComposition`中的静态方法解析json文件创建`LottieComposition lottieComposition`，这个过程中已经创建来多个`Layer`对象。
4. `lottieDrawable.setComposition(lottieComposition)`

```
void setComposition(LottieComposition composition) {
    if (getCallback() == null) {
        throw new IllegalStateException(
            "You or your view must set a Drawable.Callback before setting the co
                "gets done automatically when added to an ImageView. " +
                "Either call ImageView.setImageDrawable() before setComposition(
                "setCallback(yourView.getCallback()) first.");
    }
    clearComposition();
    this.composition = composition;
    animator.setDuration(composition.getDuration());
    setBounds(0, 0, composition.getBounds().width(), composition.getBounds().h
    buildLayersForComposition(composition);

    getCallback().invalidateDrawable(this);
}
```

先清理之前的数据，然后开始`buildLayersForComposition`，即根据`lottieComposition`建立多个`layerView`，此时已经创建好了多个Drawable，并通过List建立的为以`lottieDrawable`为根的一个drawable树。

1. `lottieAnimationView.setImageDrawable(lottieDrawable)`
2. `lottieAnimationView.playAnimation()`

```
public void playAnimation() {
    if (isAnimationLoading) {
        playAnimationWhenCompositionSet = true;
        return;
    }
    lottieDrawable.playAnimation();
}
```

直接委托给了`lottieDrawable`，`lottieDrawable`中有`private final ValueAnimator animator = ValueAnimator.ofFloat(0f, 1f);`

```
    animator.setRepeatCount(0);
    animator.setInterpolator(new LinearInterpolator());
    animator.addUpdateListener((animation) → {
        setProgress(animation.getAnimatedFraction());
    });
```

重点看下`setProgress`方法

```
public void setProgress(@FloatRange(from = 0f, to = 1f) float progress) {
  this.progress = progress;
  for (int i = 0; i < animations.size(); i++) {
    animations.get(i).setProgress(progress);
  }

  for (int i = 0; i < layers.size(); i++) {
    layers.get(i).setProgress(progress);
  }
}
```

调用了`private final List<KeyframeAnimation<?>> animations = new ArrayList<>()`的`setProgress`：

```
void setProgress(@FloatRange(from = 0f, to = 1f) float progress) {
  if (progress < getStartDelayProgress()) {
    progress = 0f;
  } else if (progress > getDurationEndProgress()) {
    progress = 1f;
  } else {
    progress = (progress - getStartDelayProgress()) / getDurationRangeProgress();
  }
  if (progress == this.progress) {
    return;
  }
  this.progress = progress;

  T value = getValue();
  for (int i = 0; i < listeners.size(); i++) {
    listeners.get(i).onValueChanged(value);
  }
}
```

在`onValueChanged`时，各个创建好的Drawable会根据需求进行重绘，达到动画的效果。

Lottie把动画从View的动效转移到了Drawable上。

Lottie的性能

可以看到Lottie把json描述的动画数据映射到Drawable之后，实现动画时用到了`ValueAnimator`，在动画更新时使用Drawable而非View，个人感觉在不需要交互时Drawable显然比View更加轻量。以下是Lottie性能的官方的说明：

> 1. 如果没有mask和mattes，那么性能和内存非常好，没有bitmap创建，大部分操作都是简单的cavas绘制。
> 2. 如果存在mattes，将会创建2～3个bitmap。bitmap在动画加载到window时被创建，被window删除时回收。所以不宜在RecyclerView中使用包涵mattes或者mask的动画，否则会

引起bitmap抖动。除了内存抖动，mattes和mask中必要的bitmap.eraseColor()和canvas.drawBitmap()也会降低动画性能。对于简单的动画，在实际使用时性能不太明显。

3. 如果在列表中使用动画，推荐使用缓存LottieAnimationView.setAnimation(String, CacheStrategy) 。