# 9.1 Flutter动画简介

在任何系统的UI框架中，动画实现的原理都是相同的，即：在一段时间内，快速地多次改变UI外观；由于人眼会产生视觉暂留，所以最终看到的就是一个“连续”的动画，这和电影的原理是一样的。我们将UI的一次改变称为一个动画帧，对应一次屏幕刷新，而决定动画流畅度的一个重要指标就是帧率FPS（Frame Per Second），即每秒的动画帧数。很明显，帧率越高则动画就会越流畅！一般情况下，对于人眼来说，动画帧率超过16FPS，就比较流畅了，超过32FPS就会非常的细腻平滑，而超过32FPS，人眼基本上就感受不到差别了。由于动画的每一帧都是要改变UI输出，所以在一个时间段内连续的改变UI输出是比较耗资源的，对设备的软硬件系统要求都较高，所以在UI系统中，动画的平均帧率是重要的性能指标，而在Flutter中，理想情况下是可以实现60FPS的，这和原生应用能达到的帧率是基本是持平的。

### Flutter中动画抽象

为了方便开发者创建动画，不同的UI系统对动画都进行了一些抽象，比如在Android中可以通过XML来描述一个动画然后设置给View。Flutter中也对动画进行了抽象，主要涉及Animation、Curve、Controller、Tween这四个角色，它们一起配合来完成一个完整动画，下面我们一一来介绍它们。

### Animation

Animation是一个抽象类，它本身和UI渲染没有任何关系，而它主要的功能是保存动画的插值和状态；其中一个比较常用的Animation类是Animation<double>。Animation对象是一个在一段时间内依次生成一个区间(Tween)之间值的类。Animation对象在整个动画执行过程中输出的值可以是线性的、曲线的、一个步进函数或者任何其他曲线函数等等，这由Curve来决定。 根据Animation对象的控制方式，动画可以正向运行（从起始状态开始，到终止状态结束），也可以反向运行，甚至可以在中间切换方向。Animation还可以生成除double之外的其他类型值，如：Animation<Color> 或Animation<Size>。在动画的每一帧中，我们可以通过Animation对象的value属性获取动画的当前状态值。

#### 动画通知

我们可以通过Animation来监听动画每一帧以及执行状态的变化，Animation有如下两个方法：

1. addListener()；它可以用于给Animation添加帧监听器，在每一帧都会被调用。帧监听器中最常见的行为是改变状态后调用setState()来触发UI重建。
2. addStatusListener()；它可以给Animation添加“动画状态改变”监听器；动画开始、结束、正向或反向（见AnimationStatus定义）时会调用状态改变的监听器。

读者在此只需要知道帧监听器和状态监听器的区别，在后面的章节中我们将会举例说明。

### Curve

动画过程可以是匀速的、匀加速的或者先加速后减速等。Flutter中通过Curve（曲线）来描述动画过程，我们把匀速动画称为线性的(Curves.linear)，而非匀速动画称为非线性的。

我们可以通过CurvedAnimation来指定动画的曲线，如：

final CurvedAnimation curve =  
 new CurvedAnimation(parent: controller, curve: Curves.easeIn);

CurvedAnimation和AnimationController（下面介绍）都是Animation<double>类型。CurvedAnimation可以通过包装AnimationController和Curve生成一个新的动画对象 ，我们正是通过这种方式来将动画和动画执行的曲线关联起来的。我们指定动画的曲线为Curves.easeIn，它表示动画开始时比较慢，结束时比较快。 [Curves](https://docs.flutter.io/flutter/animation/Curves-class.html) 类是一个预置的枚举类，定义了许多常用的曲线，下面列几种常用的：

|  |  |
| --- | --- |
| Curves曲线 | 动画过程 |
| linear | 匀速的 |
| decelerate | 匀减速 |
| ease | 开始加速，后面减速 |
| easeIn | 开始慢，后面快 |
| easeOut | 开始快，后面慢 |
| easeInOut | 开始慢，然后加速，最后再减速 |

除了上面列举的， [Curves](https://docs.flutter.io/flutter/animation/Curves-class.html) 类中还定义了许多其它的曲线，在此便不一一介绍，读者可以自行查看Curves类定义。

当然我们也可以创建自己Curve，例如我们定义一个正弦曲线：

class ShakeCurve extends Curve {  
 @override  
 double transform(double t) {  
 return math.sin(t \* math.PI \* 2);  
 }  
}

### AnimationController

AnimationController用于控制动画，它包含动画的启动forward()、停止stop() 、反向播放 reverse()等方法。AnimationController会在动画的每一帧，就会生成一个新的值。默认情况下，AnimationController在给定的时间段内线性的生成从0.0到1.0（默认区间）的数字。 例如，下面代码创建一个Animation对象（但不会启动动画）：

final AnimationController controller = new AnimationController(  
 duration: const Duration(milliseconds: 2000), vsync: this);

AnimationController生成数字的区间可以通过lowerBound和upperBound来指定，如：

final AnimationController controller = new AnimationController(   
 duration: const Duration(milliseconds: 2000),   
 lowerBound: 10.0,  
 upperBound: 20.0,  
 vsync: this  
);

AnimationController派生自Animation<double>，因此可以在需要Animation对象的任何地方使用。 但是，AnimationController具有控制动画的其他方法，例如forward()方法可以启动正向动画，reverse()可以启动反向动画。在动画开始执行后开始生成动画帧，屏幕每刷新一次就是一个动画帧，在动画的每一帧，会随着根据动画的曲线来生成当前的动画值（Animation.value），然后根据当前的动画值去构建UI，当所有动画帧依次触发时，动画值会依次改变，所以构建的UI也会依次变化，所以最终我们可以看到一个完成的动画。 另外在动画的每一帧，Animation对象会调用其帧监听器，等动画状态发生改变时（如动画结束）会调用状态改变监听器。

duration表示动画执行的时长，通过它我们可以控制动画的速度。

**注意**： 在某些情况下，动画值可能会超出AnimationController的[0.0，1.0]的范围，这取决于具体的曲线。例如，fling()函数可以根据我们手指滑动（甩出）的速度(velocity)、力量(force)等来模拟一个手指甩出动画，因此它的动画值可以在[0.0，1.0]范围之外 。也就是说，根据选择的曲线，CurvedAnimation的输出可以具有比输入更大的范围。例如，Curves.elasticIn等弹性曲线会生成大于或小于默认范围的值。

#### Ticker

当创建一个AnimationController时，需要传递一个vsync参数，它接收一个TickerProvider类型的对象，它的主要职责是创建Ticker，定义如下：

abstract class TickerProvider {  
 //通过一个回调创建一个Ticker  
 Ticker createTicker(TickerCallback onTick);  
}

Flutter应用在启动时都会绑定一个SchedulerBinding，通过SchedulerBinding可以给每一次屏幕刷新添加回调，而Ticker就是通过SchedulerBinding来添加屏幕刷新回调，这样一来，每次屏幕刷新都会调用TickerCallback。使用Ticker(而不是Timer)来驱动动画会防止屏幕外动画（动画的UI不在当前屏幕时，如锁屏时）消耗不必要的资源，因为Flutter中屏幕刷新时会通知到绑定的SchedulerBinding，而Ticker是受SchedulerBinding驱动的，由于锁屏后屏幕会停止刷新，所以Ticker就不会再触发。

通常我们会将SingleTickerProviderStateMixin添加到State的定义中，然后将State对象作为vsync的值，这在后面的例子中可以见到。

### Tween

默认情况下，AnimationController对象值的范围是[0.0，1.0]。如果我们需要构建UI的动画值在不同的范围或不同的数据类型，则可以使用Tween来添加映射以生成不同的范围或数据类型的值。例如，像下面示例，Tween生成[-200.0，0.0]的值：

final Tween doubleTween = new Tween<double>(begin: -200.0, end: 0.0);

Tween构造函数需要begin和end两个参数。Tween的唯一职责就是定义从输入范围到输出范围的映射。输入范围通常为[0.0，1.0]，但这不是必须的，我们可以自定义需要的范围。

Tween继承自Animatable<T>，而不是继承自Animation<T>，Animatable中主要定义动画值的映射规则。

下面我们看一个ColorTween将动画输入范围映射为两种颜色值之间过渡输出的例子：

final Tween colorTween =  
 new ColorTween(begin: Colors.transparent, end: Colors.black54);

Tween对象不存储任何状态，相反，它提供了evaluate(Animation<double> animation)方法，它可以获取动画当前映射值。 Animation对象的当前值可以通过value()方法取到。evaluate函数还执行一些其它处理，例如分别确保在动画值为0.0和1.0时返回开始和结束状态。

#### Tween.animate

要使用Tween对象，需要调用其animate()方法，然后传入一个控制器对象。例如，以下代码在500毫秒内生成从0到255的整数值。

final AnimationController controller = new AnimationController(  
 duration: const Duration(milliseconds: 500), vsync: this);  
Animation<int> alpha = new IntTween(begin: 0, end: 255).animate(controller);

注意animate()返回的是一个Animation，而不是一个Animatable。

以下示例构建了一个控制器、一条曲线和一个Tween：

final AnimationController controller = new AnimationController(  
 duration: const Duration(milliseconds: 500), vsync: this);  
final Animation curve =  
 new CurvedAnimation(parent: controller, curve: Curves.easeOut);  
Animation<int> alpha = new IntTween(begin: 0, end: 255).animate(curve);