

# 【Flutter】flutter\_animate 库使用

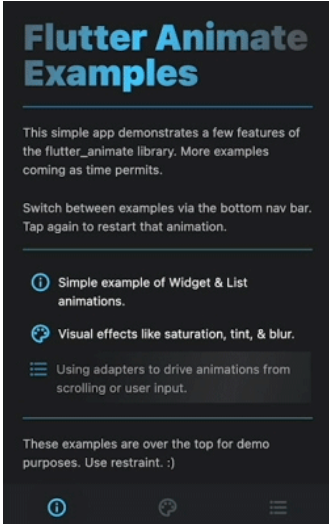
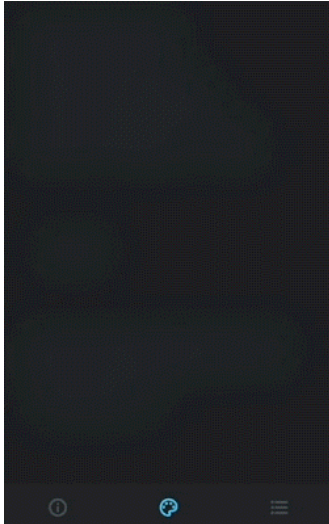
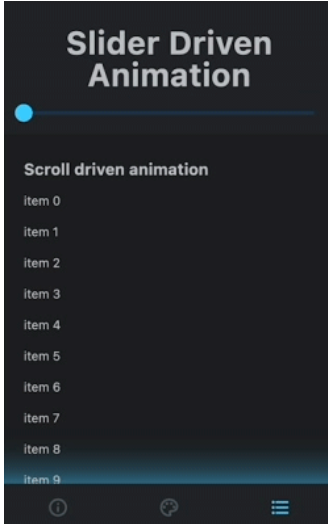
Wing\_Li

## Flutter 动画库

一个性能良好的库，使在 Flutter 中添加几乎任何类型的动画效果变得简单。

1. 预构建效果，如淡入淡出、缩放、滑动、对齐、翻转、模糊、抖动、闪烁、阴影、交叉淡入淡出、沿路径跟随和颜色效果（饱和度、颜色和色调）
2. 将动画 GLSL 片段着色器应用于小部件
3. 简化的自定义效果和动画构建器
4. 将动画与滚动、通知器或任何内容同步
5. 集成事件

所有这些功能都可以通过一个简单的统一 API 实现，无需处理 `AnimationController` 和 `StatefulWidget`。

基本动画	视觉效果	同步动画
		

上图：包含的示例应用程序。

## Duration 扩展

为 `num` 添加扩展方法，使指定持续时间更容易。例如：  
`2.seconds`、`0.1.minutes` 或 `300.ms`。

# AnimatedController 扩展

`AnimatedController` 的 `loop` 扩展方法与 `repeat` 相同，但添加了一个 `count` 参数来指定播放次数。

## 基础知识 语法

要应用效果，将目标小部件包裹在 `Animate` 中，并指定一个效果列表：

```
Animate(  
  effects: [FadeEffect(), ScaleEffect()],  
  child: Text("Hello World!"),  
)
```

它还为所有小部件添加了 `.animate()` 扩展方法，它将小部件包装在 `Animate()` 中。每个效果还会向 `Animate` 添加一个可链式调用的扩展方法，以启用简写语法：

```
Text("Hello World!").animate().fade().scale()
```

注意：在本文档中使用了简写样式，但无论使用哪种格式，所有功能均可用。

## 延迟、持续时间和曲线

效果具有可选的 `delay`、`duration` 和 `curve` 参数。效果并行运行，但可以使用 `delay` 使它们按顺序运行：

```
Text("Hello").animate()  
  .fade(duration: 500.ms)  
  .scale(delay: 500.ms) // 在淡入淡出之后运行
```

请注意，效果在整个动画的持续时间内处于“活动”状态，因此例如，同一目标上的两个淡入淡出效果可能会产生意外结果（下面详细介绍的 `SwapEffect` 可以帮助解决这个问题）。如果未指定（或为 `null`），这些值将从上一个效果继承，或者如果它是第一个效果，则从 `Animate.defaultDuration` 和 `Animate.defaultCurve` 继承：

```
Text("Hello World!").animate()  
  .fadeIn() // 使用 `Animate.defaultDuration`  
  .scale() // 从 fadeIn 继承持续时间  
  .move(delay: 300.ms, duration: 600.ms) // 使用新的持续时间在上述效果之后运行
```

```
.blurXY() // 从 move 继承延迟和持续时间
```

Animate 有自己的 delay 参数，它定义了动画开始播放之前的延迟。与 Effect 上的延迟不同，它只在动画重复时应用一次。

```
Text("Hello").animate(  
    delay: 1000.ms, // 这个延迟仅在开始时发生一次  
    onPlay: (controller) => controller.repeat(), // 循环播放  
)  
.fadeIn(delay: 500.ms) // 每次循环开始时发生此延迟
```

## 其他效果参数

大多数效果都包括 begin 和 end 参数，用于指定起始和结束值。通常情况下，这些参数是"智能"的，如果只指定一个参数，另一个参数将默认为"中性"值（即无视觉效果）。如果两个参数都未指定，则效果应使用视觉上令人愉悦的默认值。

```
// 不透明度为1是"中性"值  
Text("Hello").animate().fade() // begin=0, end=1  
Text("Hello").animate().fade(begin: 0.5) // end=1  
Text("Hello").animate().fade(end: 0.5) // begin=1
```

许多效果还具有其他影响其行为的参数。如果未指定，这些参数也应使用愉悦的默认值。

```
Text('Hello').animate().tint(color: Colors.purple)
```

## 使用 ThenEffect 进行顺序排列

ThenEffect 是一个特殊的便利性"效果"，它可以更容易地对效果进行排序。它通过建立一个新的基准时间，该时间等于上一个效果的结束时间和自己的可选 delay。所有后续效果的延迟都相对于这个新的基准时间。

在以下示例中，滑动效果将在淡入淡出效果结束后的 200 毫秒后运行。

```
Text("Hello").animate()  
    .fadeIn(duration: 600.ms)  
    .then(delay: 200.ms) // 基准时间=800ms  
    .slide()
```

## 动画列表

AnimateList 类提供了类似的功能，用于小部件列表，可以

通过指定的 interval 对每个子项的动画进行偏移：

```
Column(children: AnimateList(
  interval: 400.ms,
  effects: [FadeEffect(duration: 300.ms)],
  children: [Text("Hello"), Text("World"),
Text("Goodbye")],
))

// 或使用简写方式：
Column(
  children: [Text("Hello"), Text("World"),
Text("Goodbye")]
  .animate(interval: 400.ms).fade(duration: 300.ms),
)
```

## 共享效果

由于 Effect 实例是不可变的，因此可以重复使用它们。这使得可以轻松创建全局效果集合，在整个应用程序中使用并在一个地方更新它们变得容易。这对设计系统也很有用。

```
MyGlobalEffects.transitionIn = <Effect>[
  FadeEffect(duration: 100.ms, curve: Curves.easeOut),
  ScaleEffect(begin: 0.8, curve: Curves.easeIn)
]

// 然后：
Text('Hello').animate(effects:
MyGlobalEffects.transitionIn)
```

## 自定义效果和构建器

通过扩展 Effect，编写新的可重用效果非常简单，但是您还可以通过使用 CustomEffect、ToggleEffect 和 SwapEffect 来轻松创建一次性自定义效果。

### CustomEffect

CustomEffect 允许您构建自定义的动画效果。只需指定一个接受 context、value 和 child 的 builder 函数。child 是动画的目标小部件（可能已经包装在其他效果中）。例如，以下代码将在文本后面添加一个背景，并使其从红色渐变为蓝色：

```
Text("Hello World").animate().custom(
    duration: 300.ms,
    builder: (context, value, child) => Container(
        color: Color.lerp(Colors.red, Colors.blue, value),
        padding: EdgeInsets.all(8),
        child: child, // child 是被动动画化的 Text 小部件
    )
)
```

默认情况下，它提供一个从 0-1 的 value（尽管某些曲线可能生成超出此范围的值），该值基于当前时间、持续时间和曲线。您还可以在下面的示例中指定 begin 和 end 值。

Animate 可以创建没有子项的小部件，因此您可以将 CustomEffect 用作简化的构建器。例如，以下代码将构建一个从 10 开始倒数的文本，并淡出：

```
Animate().custom(
    duration: 10.seconds,
    begin: 10,
    end: 0,
    builder: (_, value, __) => Text(value.round()),
).fadeOut()
```

## ToggleEffect

ToggleEffect 也提供了构建器功能，但是与 double 不同，它提供了一个布尔值，即在效果结束之前为 true，在效果结束后（即持续时间之后）为 false。

```
Animate().toggle(
    duration: 2.seconds,
    builder: (_, value, __) => Text(value ? "Before" :
    "After"),
)
```

这也可用于激活"Animated"小部件，例如

AnimatedContainer，通过使用最小延迟来切换它们的值：

```
Animate().toggle(
    duration: 1.ms,
    builder: (_, value, __) => AnimatedContainer(
        duration: 1.seconds,
        color: value ? Colors.red : Colors.green,
    ),
)
```

## SwapEffect

SwapEffect 允许您在指定的时间点交换整个目标小部件：

```
Text("Before").animate()  
    .swap(duration: 900.ms, builder: (_, __) =>  
Text("After"))
```

这对于创建顺序效果也很有用，通过将目标小部件重新插入，从而有效地清除所有先前的效果：

```
text.animate().fadeOut(300.ms) // 淡出然后...  
    // 重新插入原始小部件并通过新的 Animate 渐变淡入：  
    .swap(builder: (_, child) => child.animate().fadeIn())
```

## ShaderEffect

ShaderEffect 使得将动画 GLSL 片段着色器应用于小部件变得简单。有关详细信息，请参阅文档。

```
myWidget.animate()  
    .shader(duration: 2.seconds, shader: myShader)  
    .fadeIn(duration: 300.ms) // 着色器可以与其他效果组合使用
```

## 事件和回调函数

Animate 包括以下回调函数：

- onInit：内部的 AnimationController 已经初始化
- onPlay：动画在任何 Animate.delay 之后开始播放
- onComplete：动画已完成

这些回调函数返回 AnimationController，可以用于操作动画（例如重复、反转等）。

```
Text("Horrible Pulsing Text")  
    .animate(onPlay: (controller) =>  
controller.repeat(reverse: true))  
    .fadeOut(curve: Curves.easeInOut)
```

要获得更细致的回调，可以使用 CallbackEffect 或 ListenEffect。

## CallbackEffect

CallbackEffect 允许您在动画中的任意位置添加回调函数。例如，在淡入效果进行到一半时添加回调：

```
Text("Hello").animate().fadeIn(duration: 600.ms)
  .callback(duration: 300.ms, callback: (_) =>
    print('halfway'))
```

与其他效果一样，它会继承前面效果的延迟和持续时间：

```
Text("Hello").animate().scale(delay: 200.ms, duration:
400.ms)
  .callback(callback: (_) => print('scale is done'))
```

## ListenEffect

ListenEffect 允许您注册一个回调函数，在给定的延迟、持续时间、曲线、开始和结束值时接收动画值（作为 double）。

```
Text("Hello").animate().fadeIn(curve: Curves.easeOutExpo)
  .listen(callback: (value) => print('current opacity:
$value'))
```

上面的示例有效，因为监听效果会从淡入效果继承持续时间和曲线，并且默认使用 begin=0, end=1。

## Adapters 和 Controllers

默认情况下，所有动画都由内部的 AnimationController 驱动，并根据经过的时间更新。如果需要更多控制，可以指定自己的外部 controller，或者使用 adapter。还可以设置 autoPlay=false，如果希望手动启动动画。

适配器将 AnimationController 与外部源同步。例如，ScrollAdapter 根据 ScrollController 更新动画，以便根据滚动交互运行复杂的动画。

仍然可以使用持续时间来定义动画，但是外部源必须提供 0-1 的值。

Flutter Animate 提供了一系列有用的适配器。请查看适配器了解更多信息。

## 对状态变化的反应

Animate 可以像 "Animated" 小部件（例如 AnimatedOpacity）一样对状态变化作出反应。只需正常设



置动画，但设置一个 target 值。当 target 的值发生变化时，它将自动动画到新的目标位置（其中 0 是开始，1 是结束）。

例如，与通过 setState 切换 \_over 的逻辑组合在一起，这将在鼠标悬停时淡出并缩放按钮：

```
MyButton().animate(target: _over ? 1 : 0)  
  .fade(end: 0.8).scaleXY(end: 1.1)
```

## 测试动画

在测试动画时，可以将 `Animate.restartOnHotReload=true` 设置为 true，这样每次热重载应用程序时，所有动画都将自动重新启动。

## 安装

从 [pub.dev](https://pub.dev) 获取它。