Flutter 之 动画 - AnimatedWidget & AnimatedBuilder （三十四）

maskerll

1. 基础动画

示例

```
class MSBasicAnimationRouter2 extends StatefulWidget {
  const MSBasicAnimationRouter2({Key? key}) : super(key:
key);

  @override
  State<MSBasicAnimationRouter2> createState() =>
      _MSBasicAnimationRouter2State();
}

class _MSBasicAnimationRouter2State extends
State<MSBasicAnimationRouter2>
    with SingleTickerProviderStateMixin {
  late AnimationController _controller;
  late var _curveAnimation;
  late var _valueAnimation;
  @override
  void initState() {
    super.initState();
    _controller =
        AnimationController(vsync: this, duration:
```

```
Duration(seconds: 5));
    _curveAnimation =
        CurvedAnimation(parent: _controller, curve:
Curves.bounceIn);
    _valueAnimation = Tween(begin: 0.0, end:
300.0).animate(_curveAnimation);

    _controller.addListener(() {
      setState(() {});
    });

    _controller.forward();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Image.asset("assets/images/fengjing4.png",
            width: _valueAnimation.value,
            height: _valueAnimation.value,
            fit: BoxFit.cover),
      ),
    );
  }

  @override
  void dispose() {
```

```
    super.dispose();
    _controller.dispose();
  }
}
```

33.gif

上面代码中 addListener() 函数调用了 setState()，所以每次动画生成一个新的数字时，当前帧被标记为脏 (dirty)，这会导致 widget 的 build() 方法再次被调用，而在 build() 中，改变 Image 的宽高，因为它的高度和宽度现在使用的是 animation.value ，所以就会逐渐放大。值得注意的是动画完成时要释放控制器 (调用 dispose() 方法) 以防止内存泄漏

上面的代码中，必须在 addListener() 中调用 setState()，动画才能正常执行，那能不能不调用 setState() 动画又能执行呢?

Flutter 中提供了 AnimatedWidget 和 AnimatedBuilder 两种

动画方式

2. AnimatedWidget 重构

AnimatedWidget类封装了调用setState()的细节，并允许我们将 widget 分离出来

AnimatedWidget 将需要执行动画的Widget放到一个AnimationWidget中的build方法中进行返回

重构后的代码如下：

```
class MSAnimatedImageWidget extends AnimatedWidget {
  MSAnimatedImageWidget({required Animation<double>
listenable})
      : super(listenable: listenable);
  @override
  Widget build(BuildContext context) {
    print("MSAnimatedImageWidget build");
    final anim = listenable as Animation<double>;
    return Image.asset("assets/images/fengjing4.png",
        width: anim.value, height: anim.value, fit:
BoxFit.cover);
  }
}
```

完整代码

移除 setState

```
```

```
class MSAnimationWidgetRouter2 extends StatefulWidget {
  const MSAnimationWidgetRouter2({Key? key}) : super(key:
key);

  @override
  State<MSAnimationWidgetRouter2> createState() =>
      _MSAnimationWidgetRouter2State();
}

class _MSAnimationWidgetRouter2State extends
State<MSAnimationWidgetRouter2>
    with SingleTickerProviderStateMixin {
  late AnimationController _controller;
  late var _curveAnimation;
  late var _valueAnimation;
  @override
  void initState() {
    super.initState();
    _controller =
        AnimationController(vsync: this, duration:
Duration(seconds: 5));
    _curveAnimation =
        CurvedAnimation(parent: _controller, curve:
Curves.bounceIn);
    _valueAnimation = Tween(begin: 0.0, end:
300.0).animate(_curveAnimation);
    _controller.forward();
  }
```

```
  @override
  Widget build(BuildContext context) {
    print("_MSAnimationWidgetRouter2State build");
    return Scaffold(
      body: Center(
        child: MSAnimatedImageWidget(
          listenable: _valueAnimation,
        ),
      ),
    );
  }

  @override
  void dispose() {
    super.dispose();
    _controller.dispose();
  }
}


class MSAnimatedImageWidget extends AnimatedWidget {
  MSAnimatedImageWidget({required Animation<double>
listenable})
      : super(listenable: listenable);
  @override
  Widget build(BuildContext context) {
    print("MSAnimatedImageWidget build");
```

```
    final anim = listenable as Animation<double>;
    return Image.asset("assets/images/fengjing4.png",
        width: anim.value, height: anim.value, fit:
BoxFit.cover);
  }
}
```

在打印日志中，我们发现 _MSAnimationWidgetRouter2State 只会 build 一次，MSAnimatedImageWidget 会一直 build，直到动画停止。

3.AnimatedBuilder 重构

用 AnimatedWidget 可以从动画中分离出 widget，而动画的渲染过程（即设置宽高）仍然在 AnimatedWidget 中，假设如果我们再添加一个 widget 透明度变化的动画，那么我们需要再实现一个 AnimatedWidget，这样不是很优雅，如果我们能把渲染过程也抽象出来，那就会好很多，而 AnimatedBuilder 正是将渲染逻辑分离出来, 上面的 build 方法中的代码可以改为：

```
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: AnimatedBuilder(
          animation: _valueAnimation,
          builder: (context, child) {
            return SizedBox(
```

```
              width: _valueAnimation.value,
              height: _valueAnimation.value,
              child: child,
            );
          },
          child: Image.asset("assets/images/fengjing4.png",
fit: BoxFit.cover),
        ),
      ),
    );
  }
```

完整代码

```
class MSAnimationBuilderRouter2 extends StatefulWidget {
  const MSAnimationBuilderRouter2({Key? key}) : super(key:
key);

  @override
  State<MSAnimationBuilderRouter2> createState() =>
      _MSAnimationBuilderRouter2State();
}

class _MSAnimationBuilderRouter2State extends
State<MSAnimationBuilderRouter2>
    with SingleTickerProviderStateMixin {
  late AnimationController _controller;
  late var _curveAnimation;
```

```
  late var _valueAnimation;
  @override
  void initState() {
    super.initState();
    _controller =
        AnimationController(vsync: this, duration:
Duration(seconds: 5));
    _curveAnimation =
        CurvedAnimation(parent: _controller, curve:
Curves.bounceIn);
    _valueAnimation = Tween(begin: 0.0, end:
300.0).animate(_curveAnimation);
    _controller.forward();
  }

  @override
  Widget build(BuildContext context) {
    print("_MSAnimationBuilderRouter2State build");
    return Scaffold(
      body: Center(
        child: AnimatedBuilder(
          animation: _valueAnimation,
          builder: (context, child) {
            print("AnimatedBuilder build");
            return SizedBox(
              width: _valueAnimation.value,
              height: _valueAnimation.value,
              child: child,
```

```
            );
          },
          child: Image.asset("assets/images/fengjing4.png",
fit: BoxFit.cover),
        ),
      ),
    );
  }

  @override
  void dispose() {
    super.dispose();
    _controller.dispose();
  }
}
```

在打印日志中，我们发现_MSAnimationBuilderRouter2State只会build一次，AnimatedBuilder会一直build，直到动画停止。

上面的代码中有一个迷惑的问题是，child看起来像被指定了两次。但实际发生的事情是：将外部引用child传递给AnimatedBuilder后，AnimatedBuilder再将其传递给匿名构造器， 然后将该对象用作其子对象。最终的结果是AnimatedBuilder返回的对象插入到 widget 树中。

AnimatedBuilder会带来三个好处：

- 不用显式的去添加帧监听器，然后再调用setState()

了，这个好处和 AnimatedWidget 是一样的。

- 更好的性能：因为动画每一帧需要构建的 widget 的范围缩小了，如果没有 builder，setState() 将会在父组件上下文中调用，这将会导致父组件的 build 方法重新调用；而有了 builder 之后，只会导致动画 widget 自身的 build 重新调用，避免不必要的 rebuild。

- 通过 AnimatedBuilder 可以封装常见的过渡效果来复用动画。

下面我们通过封装一个 GrowTransition 来说明，它可以对子 widget 实现放大动画：

```
class MSGrowTransition extends StatelessWidget {
  const MSGrowTransition({Key? key, required this.anim,
this.child})
      : super(key: key);

  final Animation<double> anim;
  final Widget? child;

  @override
  Widget build(BuildContext context) {
    return AnimatedBuilder(
      animation: anim,
```

```
      builder: (ctx, child) {
        return SizedBox(
          width: anim.value,
          height: anim.value,
          child: child,
        );
      },
      child: child,
    );
  }
}
```

最初的示例就可以改为

```
...
  Widget build(BuildContext context) {
    print("_MSAnimationBuilderRouter2State build");
    return Scaffold(
      body: Center(
        child: MSGrowTransition(
          anim: _valueAnimation,
          child: Image.asset("assets/images/fengjing4.png",
fit: BoxFit.cover),
        ),
      ),
    );
  }
...
```

Flutter中正是通过这种方式封装了很多动画，如：

FadeTransition、ScaleTransition、SizeTransition等，很多时候都可以复用这些预置的过渡类

4. 动画状态监听 案例

可以通过`Animation`的`addStatusListener()`方法来添加动画状态改变监听器。Flutter中，有四种动画状态，在`AnimationStatus`枚举类中定义。

枚举值	含义
`dismissed`	动画在起始点停止
`forward`	动画正在正向执行
`reverse`	动画正在反向执行
`completed`	动画在终点停止

```
class MSAnimationStateRouter extends StatefulWidget {
  const MSAnimationStateRouter({Key? key}) : super(key:
key);

  @override
  State<MSAnimationStateRouter> createState() =>
_MSAnimationStateRouterState();
}
```

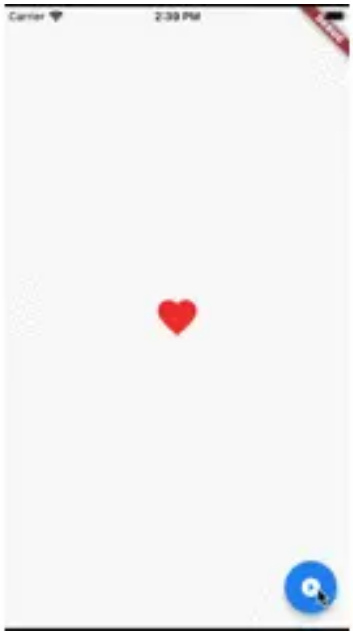
```
class _MSAnimationStateRouterState extends
State<MSAnimationStateRouter>
    with SingleTickerProviderStateMixin {
  late AnimationController _controller;
  late Animation<double> _curveAnimation;
  late Animation<double> _valueAnimation;
  @override
  void initState() {
    super.initState();
    _controller =
        AnimationController(vsync: this, duration:
Duration(seconds: 3));
    _curveAnimation =
        CurvedAnimation(parent: _controller, curve:
Curves.linear);
    _valueAnimation = Tween(begin: 50.0, end:
150.0).animate(_curveAnimation);

    _controller.addStatusListener((status) {
      if (status == AnimationStatus.dismissed) {
        _controller.forward();
      } else if (status == AnimationStatus.completed) {
        _controller.reverse();
      }
    });
  }

  @override
```

```
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: AnimatedBuilder(
          animation: _controller,
          builder: (ctx, child) {
            return Icon(Icons.favorite,
                color: Colors.red, size:
_valueAnimation.value);
          },
        ),
      ),
      floatingActionButton: FloatingActionButton(
        child: Icon(Icons.play_circle_fill),
        onPressed: () {
          if (_controller.isAnimating) {
            _controller.stop();
          } else {
            if (_controller.status ==
AnimationStatus.dismissed ||
                _controller.status ==
AnimationStatus.forward) {
              _controller.forward();
            } else {
              _controller.reverse();
            }
          }
        },
```

```
      ),
    );
  }
}
```



35.gif

https://book.flutterchina.club/chapter9/animation_structure.html