我什么时候应该使用 AnimatedBuilder 或 AnimatedWidget

whqfor

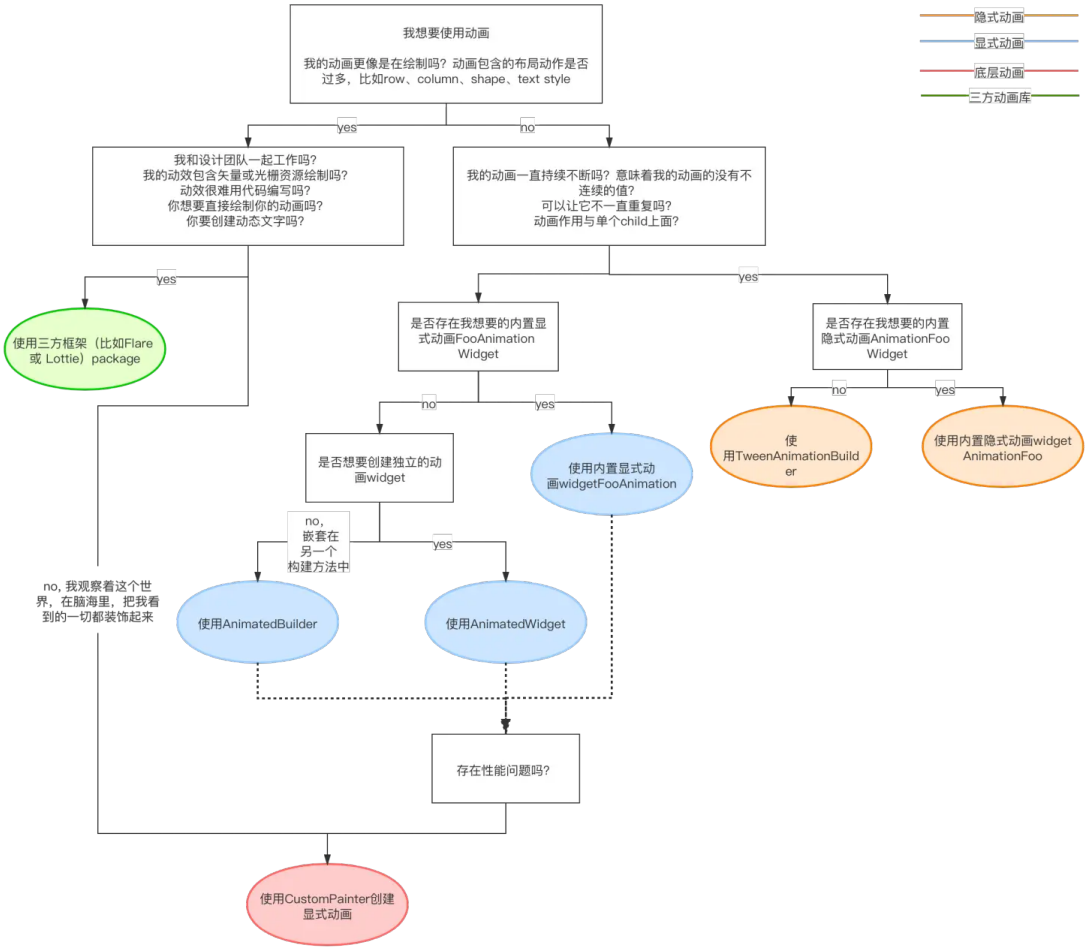
我们知道当你乘坐飞机飞行的时候你有很多选择，这里我想表达的是在 Flutter 中选择动画，首先感谢你选择使用 `AnimatedBuilder` 和 `AnimatedWidget`，等等，什么，还没有使用？Flutter 有很多不同的动画 widget，但是与商业航空公司不一样的是，flutter 中的每种类型的 widget 都有自己的适用场景。当然，你可以使用两种不同的方式来完成一样的动画，但是使用适当的 animation widget 来完成这项工作，将会更加轻松。

这篇文章介绍了和其他动画 widget 对比，你为什么可能需要使用 `AnimatedBuilder` 和 `AnimatedWidget`，以及如何使用它们，假设你想向你的 APP 中添加动画。本文是该系列文章的一部分，逐步介绍了可能希望使用的各种类型的动画 widget。你想要特定动画重复执行几次，或者想要暂停、开始以响应某些事件，比如手指点击，由于您的动画需要重复或停止、开始，因此你将需要使用显式动画。

顺便说一下，Flutter 有两大类型动画：显式和隐式。对于显式动画，你需要一个 animation controller，对于隐式动画则不需要。在上篇关于使用内置显示动画的文章，我们介绍了 animation controller，假如你想要了解更多关于此的内容，

请先查看那篇文章。

到此，如果你确定使用显式动画，有很多显式动画供您选择，这些类通常命名为`FooTransition`，`Foo`是您想要设置的动画的属性名称，我建议先了解一下是否可以使用其中的一个widget来实现你的需求，然后再深入了解`AnimatedBuilder`和`AnimatedWidget`。有很多效果很棒的widget供您选择，包括旋转、位移、对齐、淡入淡出、文本样式等，另外你可以组合这些Widget，这样就可以同时进行旋转和淡入淡出效果。但是，如果这些内置的Widget不能满足你的需求，那么就是时机使用`AnimatedBuilder`和`AnimatedWidget`了。

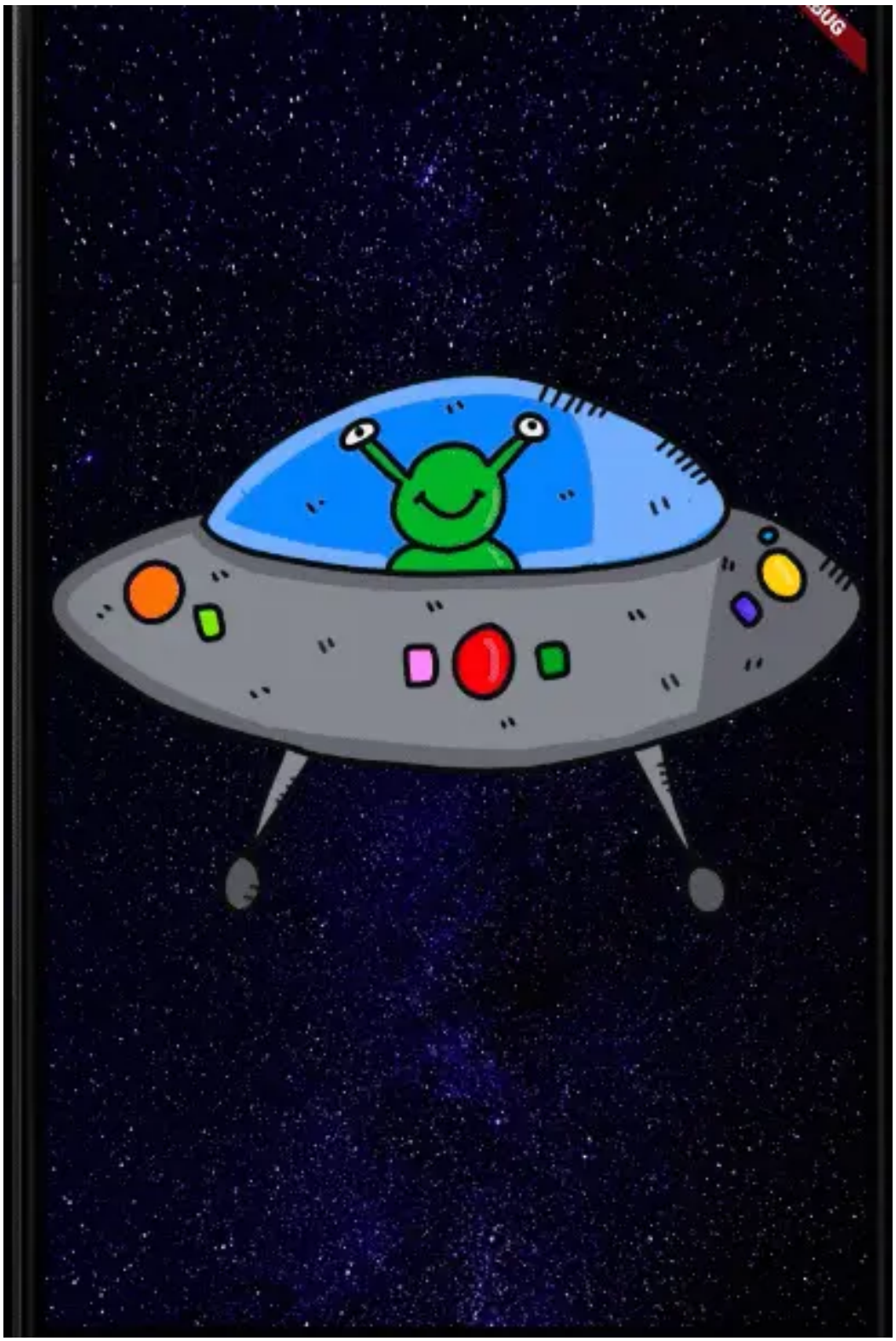


whichanimation.png

这是用于了解使用哪种动画的流程图，本文重点介绍底部的两个蓝色部分，AnimatedBuilder and AnimatedWidget。

特别的例子

为了使以上内容更加具体，让我们来看一个具体的场景：我想编写一个带有外星飞船的 APP，这个飞船有一个光柱动画。



我绘制了一个渐变色的飞船光束，渐变色从正中心向外逐步黄色变为透明，然后，我使用路径裁剪（path clipper）从该

渐变创建了一个光束的形状。

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
      body: MyHomePage(),
    ));
  }
}

class MyHomePage extends StatelessWidget {
  final Image starsBackground = Image.asset(
    'assets/milky-way.jpg',
  );
  final Image ufo = Image.asset('assets/ufo.png');
  @override
  Widget build(BuildContext context) {
    return Stack(
      alignment: AlignmentDirectional.center,
      children: <Widget>[
        starsBackground,
```

```
        ClipPath(
          clipper: const BeamClipper(),
          child: Container(
            height: 1000,
            decoration: BoxDecoration(
              gradient: RadialGradient(
                radius: 1.5,
                colors: [
                  Colors.yellow,
                  Colors.transparent,
                ],
              ),
            ),
          ),
        ),
        ufo,
      ],
    );
  }
}

class BeamClipper extends CustomClipper<Path> {
  const BeamClipper();

  @override
  getClip(Size size) {
    return Path()
      ..lineTo(size.width / 2, size.height / 2)
```

```
      ..lineTo(size.width, size.height)
      ..lineTo(0, size.height)
      ..lineTo(size.width / 2, size.height / 2)
      ..close();
  }

  /// Return false always because we always clip the same
area.
  @override
  bool shouldReclip(CustomClipper oldClipper) => false;
}
```

我想要创建一个`光束降落`的动画，从该渐变的中心开始，并使其重复。这意味着我需要创建显式动画，不幸的是，没有内置的显式动画来为漏斗形渐变设置动画，但是你知道我们有...`AnimatedBuilder`和`AnimatedWidget`可以解决这个问题！

AnimatedBuilder

为了制作光束动画，我将把这段渐变代码包裹在`AnimatedBuilder` widget中。当`AnimatedBuilder`被调用的时候，包含在builder函数中渐变代码也将被调用。

接下来我需要添加一个controller来驱动动画，controller将会提供`AnimatedBuilder`用来逐帧绘制所需要的值。如你在之前的文章里看到的，我混入（mix in）了`SingleTickerProviderStateMixin`类，并在`initState`而不是`build`方法中初始化了controller实例对象，因为我不想多次

创建controller--我想要它为动画的每一帧提供新的值！因为我在`initState`中创建了一个新的对象，所以我也添加了一个`dispose`方法，用来告知Flutter，当不再有父节点widget显示在屏幕上的时候，可以销毁controller。
然后，我将controller传递给`AnimatedBuilder`，动画按照预期运行啦！

```
class MyHomePage extends StatefulWidget {
  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage>
    with SingleTickerProviderStateMixin {
  final Image starsBackground = Image.asset(
    'assets/milky-way.jpg',
  );
  final Image ufo = Image.asset('assets/ufo.png');
  AnimationController _animation;

  @override
  void initState() {
    super.initState();
    _animation = AnimationController(
      duration: const Duration(seconds: 5),
      vsync: this,
    )..repeat();
```

```
}

@override
Widget build(BuildContext context) {
  return Stack(
    alignment: AlignmentDirectional.center,
    children: <Widget>[
      starsBackground,
      AnimatedBuilder(
        animation: _animation,
        builder: (_, __) {
          return ClipPath(
            clipper: const BeamClipper(),
            child: Container(
              height: 1000,
              decoration: BoxDecoration(
                gradient: RadialGradient(
                  radius: 1.5,
                  colors: [
                    Colors.yellow,
                    Colors.transparent,
                  ],
                  stops: [0, _animation.value],
                ),
              ),
            ),
          );
        },
```

```
        ),
        ufo,
      ],
    );
  }

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

class BeamClipper extends CustomClipper<Path> {
  const BeamClipper();

  @override
  getClip(Size size) {
    return Path()
      ..lineTo(size.width / 2, size.height / 2)
      ..lineTo(size.width, size.height)
      ..lineTo(0, size.height)
      ..lineTo(size.width / 2, size.height / 2)
      ..close();
  }

  /// Return false always because we always clip the same
area.
```

```
  @override
  bool shouldReclip(CustomClipper oldClipper) => false;
}
```

你可能还记得在TweenAnimationBuilder一文中，我们提到使用child参数来进行性能优化，我们在`AnimatedBuilder`中也可以这样做。基本上，如果我们在动画中有从来没改变过的对象，则可以提前构建他们，然后将它传递到`AnimatedBuilder`中。

在这个例子中，有一种更好的实现方式来做同样的事情：给`BeamClipper`设置一个`const`构造函数，并且仅仅设置了`const`。这样只需要少量的代码，这个对象将会在编译期创建，使构建更快速。当然，有时你会编写一些没有const构造函数的代码，这种情况对与使用可选child参数来说是个很好的应用场景。

AnimatedWidget

到此，我们创建了自己的动画，但是包含`AnimatedBuilder`的构建函数代码量有点大，假如你的构建方法开始变的有点难以阅读，是时候重构代码了。

你可以将`AnimatedBuilder`代码提取到单独的Widget中，但是这样的话，你的构建方法中将会嵌套另一个构建方法，看起来有点丑陋。取而代之的是，你可以通过继承自`AnimatedWidget`创建一个新的Widget来完成相同的动画。我将我的Widget命名为`BeamTransition`，与`FooTransition`显示

动画的命名习惯一致。我将 animation controller 传递给 `BeamTransition`，并重用了 `AnimatedBuilder` 构造函数的主体代码。

```
class MyHomePage extends StatefulWidget {
  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage>
    with SingleTickerProviderStateMixin {
  final Image starsBackground = Image.asset(
    'assets/milky-way.jpg',
  );
  final Image ufo = Image.asset('assets/ufo.png');
  AnimationController _animation;

  @override
  void initState() {
    super.initState();
    _animation = AnimationController(
      duration: const Duration(seconds: 5),
      vsync: this,
    )..repeat();
  }

  @override
  Widget build(BuildContext context) {
```

```
    return Stack(
      alignment: AlignmentDirectional.center,
      children: <Widget>[
        starsBackground,
        BeamTransition(animation: _animation),
        ufo,
      ],
    );
  }

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

class BeamTransition extends AnimatedWidget {
  BeamTransition({Key key, Animation<double> animation})
      : super(key: key, listenable: animation);
  @override
  Widget build(BuildContext context) {
    final Animation<double> animation = listenable;
    return ClipPath(
      clipper: const BeamClipper(),
      child: Container(
        height: 1000,
        decoration: BoxDecoration(
```

```
          gradient: RadialGradient(
            radius: 1.5,
            colors: [
              Colors.yellow,
              Colors.transparent,
            ],
            stops: [0, animation.value],
          ),
        ),
      ),
    );
  }
}
```

就像`AnimatedBuilder`一样，如果可能的话，我将添加 child 参数到我的 widget 中，以便进行性能优化，因为它可以提前而不是每次进行动画时进行构建。顺带提醒一下，在此例子中，将`BeamClipper`采用`const`构造声明是最好的方式。

那么，我到底该用哪个呐?

我们刚刚看到了，当你无法找到内置显式动画想要实现你想要的效果时，`AnimatedBuilder`和`AnimatedWidget`都可以用来实现相同效果的显式动画，那么，你该用哪一个呐？这是一个个人偏好问题，一般来说我建议制作独立的 widget，每个 widget 负责单独的功能--在这个例子中是动画。

绝大多数时，我都赞成使用`AnimatedWidget`，但是如果你创建 animation controller 的父节点 Widget 非常简单，那么为你的

动画创建一个独立的 Widget 可能会引入太多额外的代码，这种情况，`AnimatedBuilder` 是你的首选。

这里有这篇文章的视频版本，如果你更喜欢视频，点击观看。

系列文章：

视频	对应文章（英文原文）	对应文章（中文翻译）
如何在 Flutter 中选择合适的动画 Widget　在 Flutter 中使用动画的正确选择	How to Choose Which Flutter Animation Widget is Right for You?	【已翻译】链接
隐式动画基础	Flutter animation basics with implicit animations	【已翻译】链接
使用 TweenAnimationBuilder 创建独特的隐式动画	Custom Implicit Animations in Flutter...with TweenAnimationBuilder	【已翻译】链接

使用内置显式动画	Directional animations with built-in explicit animations	【已翻译】链接
通过 AnimatedBuilder 和 AnimatedWidget 创建一个自定义动画	When should I useAnimatedBuilder or AnimatedWidget?	【已翻译】链接
深入理解动画	Animation deep dive	【已翻译】链接