



Flutter实现动画

原创 coderwhy coderwhy

对于一个前端的App来说，添加适当的动画，可以给用户更好的体验和视觉效果。所以无论是原生的iOS或Android，还是前端开发中都会提供完成某些动画的API。

Flutter有自己的渲染闭环，我们当然可以给它提供一定的数据模型，来让它帮助我们实现对应的动画效果。

一. 动画API认识

动画其实是我们通过某些方式（比如对象，Animation对象）给Flutter引擎提供不同的值，而Flutter可以根据我们提供的值，给对应的Widget添加顺滑的动画效果。

针对动画这个章节，我打算先理清楚他们的API关系和作用，再来讲解如何利用这些API来实现不同的动画效果。

1.1. Animation

在Flutter中，实现动画的核心类是Animation，Widget可以直接将这些动画合并到自己的build方法中来读取它们的当前值或者监听它们的状态变化。

我们一起来看一下Animation这个类，它是一个抽象类：

- addListener方法
 - 每当动画的状态值发生变化时，动画都会通知所有通过 `addListener` 添加的监听器。
 - 通常，一个正在监听动画的 `state` 对象会调用自身的 `setState` 方法，将自身传入这些监听器的回调函数来通知 widget 系统需要根据新状态值进行重新构建。
- addStatusListener
 - 当动画的状态发生变化时，会通知所有通过 `addStatusListener` 添加的监听器。
 - 通常情况下，动画会从 `dismissed` 状态开始，表示它处于变化区间的开始点。
 - 举例来说，从 0.0 到 1.0 的动画在 `dismissed` 状态时的值应该是 0.0。
 - 动画进行的下一状态可能是 `forward` （比如从 0.0 到 1.0）或者 `reverse` （比如从 1.0 到 0.0）。
 - 最终，如果动画到达其区间的结束点（比如 1.0），则动画会变成 `completed` 状态。

```

abstract class Animation<T> extends Listenable implements ValueListenable<T> {
  const Animation();

  // 添加动画监听器
  @override
  void addListener(VoidCallback listener);

  // 移除动画监听器
  @override
  void removeListener(VoidCallback listener);

  // 添加动画状态监听器
  void addStatusListener(AnimationStatusListener listener);

  // 移除动画状态监听器
  void removeStatusListener(AnimationStatusListener listener);

  // 获取动画当前状态
  AnimationStatus get status;

  // 获取动画当前的值
  @override
  T get value;
}

```

1.2. AnimationController

Animation是一个抽象类，并不能用来直接创建对象实现动画的使用。

AnimationController是Animation的一个子类，实现动画通常我们需要创建AnimationController对象。

- AnimationController会生成一系列的值，默认情况下值是0.0到1.0区间的值；

除了上面的监听，获取动画的状态、值之外，AnimationController还提供了对动画的控制：

- forward：向前执行动画
- reverse：方向播放动画
- stop：停止动画

AnimationController的源码：

```

class AnimationController extends Animation<double>
  with AnimationEagerListenerMixin, AnimationLocalListenersMixin, AnimationLocalStatusListenersMixin {
  AnimationController({
    // 初始化值
    double value,
    // 动画执行的时间
    this.duration,
    // 反向动画执行的时间
    this.reverseDuration,
    // 最小值
    this.lowerBound = 0.0,
    // 最大值
    this.upperBound = 1.0,
    // 刷新率ticker的回调（看下面详细解析）
    @required TickerProvider vsync,
  })
}

```

AnimationController有一个必传的参数vsync，它是什么呢？

- 之前我讲过关于Flutter的渲染闭环，Flutter每次渲染一帧画面之前都需要等待一个vsync信号。
- 这里也是为了监听vsync信号，当Flutter开发的应用程序不再接受同步信号时（比如锁屏或退到后台），那么继续执行动画会消耗性能。

- 这个时候我们设置了Ticker，就不会再出发动画了。
- 开发中比较常见的是将SingleTickerProviderStateMixin混入到State的定义中。

1.3. CurvedAnimation

CurvedAnimation也是Animation的一个实现类，它的目的是为了给AnimationController增加动画曲线：

- CurvedAnimation可以将AnimationController和Curve结合起来，生成一个新的Animation对象

```
class CurvedAnimation extends Animation<double> with AnimationWithParentMixin<double> {
  CurvedAnimation({
    // 通常传入一个AnimationController
    @required this.parent,
    // Curve类型的对象
    @required this.curve,
    this.reverseCurve,
  });
}
```

Curve类型的对象有一些常量Curves（和Color类型有一些Colors是一样的），可以供我们直接使用：

- 对应值的效果，可以直接查看官网（有对应的gif效果，一目了然）
- <https://api.flutter.dev/flutter/animation/Curves-class.html>

官方也给出了自己定义Curve的一个示例：

```
import 'dart:math';

class ShakeCurve extends Curve {
  @override
  double transform(double t) => sin(t * pi * 2);
}
```

1.4. Tween

默认情况下，AnimationController动画生成的值所在区间是0.0到1.0

- 如果希望使用这个以外的值，或者其他的数据类型，就需要使用Tween

Tween的源码：

- 源码非常简单，传入两个值即可，可以定义一个范围。

```
class Tween<T extends dynamic> extends Animatable<T> {
  Tween({ this.begin, this.end });
}
```

Tween也有一些子类，比如ColorTween、BorderTween，可以针对动画或者边框来设置动画的值。

Tween.animate

要使用Tween对象，需要调用Tween的animate()方法，传入一个Animation对象。

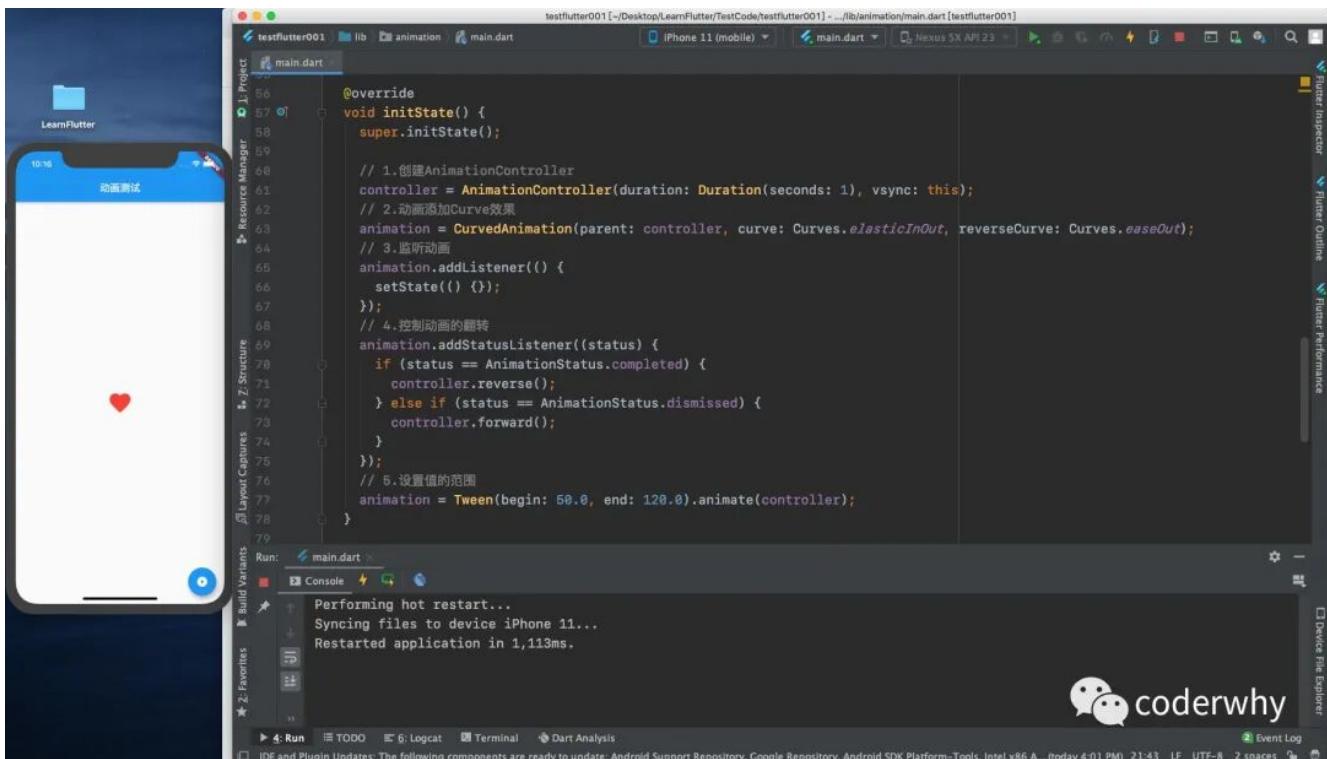
二. 动画案例练习

2.1. 动画的基本使用

我们来完成一个案例：

- 点击案例后执行一个心跳动画，可以反复执行

- 再次点击可以暂停和重新开始动画



动画效果

```

class HYHomePage extends StatelessWidget {
    final GlobalKey<_AnimationDemo01State> demo01Key = GlobalKey();

    @override
    Widget build(BuildContext context) {
        return Scaffold(
            appBar: AppBar(
                title: Text("动画测试"),
            ),
            body: AnimationDemo01(key: demo01Key),
            floatingActionButton: FloatingActionButton(
                child: Icon(Icons.play_circle_filled),
                onPressed: () {
                    if (!demo01Key.currentState.controller.isAnimating) {
                        demo01Key.currentState.controller.forward();
                    } else {
                        demo01Key.currentState.controller.stop();
                    }
                },
            ),
        );
    }
}

class AnimationDemo01 extends StatefulWidget {
    AnimationDemo01({Key key}): super(key: key);

    @override
    _AnimationDemo01State createState() => _AnimationDemo01State();
}

class _AnimationDemo01State extends State<AnimationDemo01> with SingleTickerProviderStateMixin {
    AnimationController controller;
    Animation<double> animation;

    @override
    void initState() {
        super.initState();

        // 1. 创建AnimationController
        controller = AnimationController(duration: Duration(seconds: 1), vsync: this);
    }
}

```

```

// 2.动画添加Curve效果
animation = CurvedAnimation(parent: controller, curve: Curves.elasticInOut, reverseCurve: Curves.easeOut);
// 3.监听动画
animation.addListener(() {
  setState(() {});
});
// 4.控制动画的翻转
animation.addStatusListener((status) {
  if (status == AnimationStatus.completed) {
    controller.reverse();
  } else if (status == AnimationStatus.dismissed) {
    controller.forward();
  }
});
// 5.设置值的范围
animation = Tween(begin: 50.0, end: 120.0).animate(controller);
}

@Override
Widget build(BuildContext context) {
  return Center(
    child: Icon(Icons.favorite, color: Colors.red, size: animation.value,),
  );
}

@Override
void dispose() {
  controller.dispose();
  super.dispose();
}
}

```

2.2. AnimatedWidget

在上面的代码中，我们必须监听动画值的改变，并且改变后需要调用`setState`，这会带来两个问题：

- 1.执行动画必须包含这部分代码，代码比较冗余
- 2.调用`setState`意味着整个State类中的`build`方法就会被重新`build`

如何可以优化上面的操作呢？`AnimatedWidget`

创建一个Widget继承自`AnimatedWidget`：

```

class IconAnimation extends AnimatedWidget {
  IconAnimation(Animation animation): super(listenable: animation);

  @override
  Widget build(BuildContext context) {
    Animation animation = listenable;
    return Icon(Icons.favorite, color: Colors.red, size: animation.value,);
  }
}

```

修改`_AnimationDemo01State`中的代码：

```
class _AnimationDemo01State extends State<AnimationDemo01> with SingleTickerProviderStateMixin {
    AnimationController controller;
    Animation<double> animation;

    @override
    void initState() {
        super.initState();

        // 1. 创建AnimationController
        controller = AnimationController(duration: Duration(seconds: 1), vsync: this);

        // 2. 动画添加Curve效果
        animation = CurvedAnimation(parent: controller, curve: Curves.elasticInOut, reverseCurve: Curves.easeOut);

        // 3. 监听动画
        // 4. 控制动画的翻转
        animation.addStatusListener((status) {
            if (status == AnimationStatus.completed) {
                controller.reverse();
            } else if (status == AnimationStatus.dismissed) {
                controller.forward();
            }
        });
        // 5. 设置值的范围
        animation = Tween(begin: 50.0, end: 120.0).animate(controller);
    }

    @override
    Widget build(BuildContext context) {
        return Center(
            child: IconAnimation(animation),
        );
    }

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

2.3. AnimatedBuilder

Animated是不是最佳的解决方案呢?

- 1. 我们每次都要新建一个类来继承自AnimatedWidget
- 2. 如果我们的动画Widget有子Widget, 那么意味着它的子Widget也会重新build

如何可以优化上面的操作呢? AnimatedBuilder

```
class _AnimationDemo01State extends State<AnimationDemo01> with SingleTickerProviderStateMixin {
    AnimationController controller;
    Animation<double> animation;

    @override
    void initState() {
        super.initState();

        // 1. 创建AnimationController
        controller = AnimationController(duration: Duration(seconds: 1), vsync: this);

        // 2. 动画添加Curve效果
        animation = CurvedAnimation(parent: controller, curve: Curves.elasticInOut, reverseCurve: Curves.easeOut);

        // 3. 监听动画
        // 4. 控制动画的翻转
        animation.addStatusListener((status) {
            if (status == AnimationStatus.completed) {
                controller.reverse();
            } else if (status == AnimationStatus.dismissed) {
                controller.forward();
            }
        });
        // 5. 设置值的范围
        animation = Tween(begin: 50.0, end: 120.0).animate(controller);
    }

    @override
    Widget build(BuildContext context) {
        return Center(
            child: AnimatedBuilder(
                animation: animation,
                builder: (ctx, child) {
                    return Icon(Icons.favorite, color: Colors.red, size: animation.value,);
                },
            ),
        );
    }

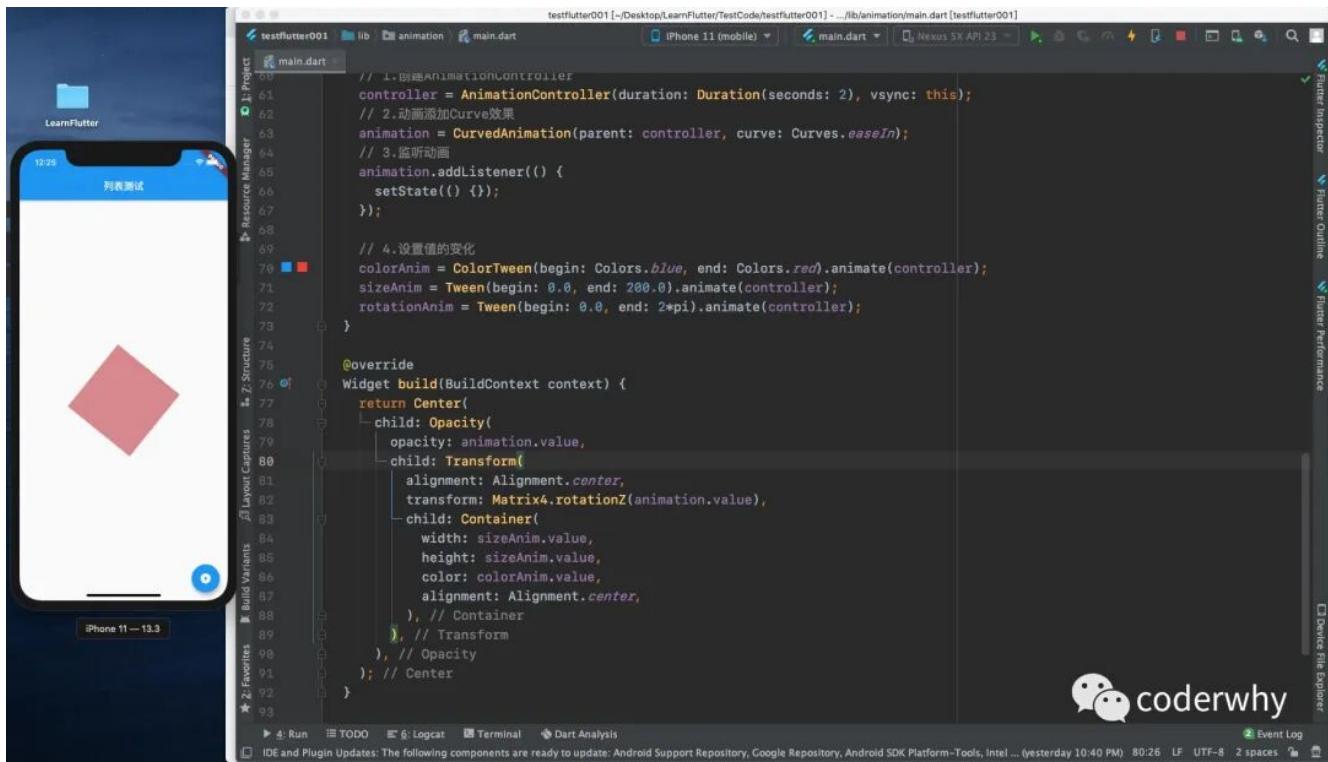
    @override
    void dispose() {
        controller.dispose();
        super.dispose();
    }
}
```

三. 其它动画补充

3.1. 交织动画

案例说明：

- 点击floatingActionButton执行动画
- 动画集合了透明度变化、大小变化、颜色变化、旋转动画等；
- 我们这里是通过多个Tween生成了多个Animation对象；



交织动画

```
import 'dart:math';

import 'package:flutter/material.dart';

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


class MyApp extends StatelessWidget {
    // This widget is the root of your application.

    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            title: 'Flutter Demo',
            theme: ThemeData(
                primarySwatch: Colors.blue, splashColor: Colors.transparent),
            home: HYHomePage(),
        );
    }
}

class HYHomePage extends StatelessWidget {
    final GlobalKey<_AnimationDemo01State> demo01Key = GlobalKey();

    @override
    Widget build(BuildContext context) {
        return Scaffold(
            appBar: AppBar(
                title: Text("列表测试"),
            ),
            body: AnimationDemo01(key: demo01Key),
            floatingActionButton: FloatingActionButton(
                child: Icon(Icons.play_circle_filled),
                onPressed: () {
                    demo01Key.currentState.controller.forward();
                },
            ),
        );
    }
}

class AnimationDemo01 extends StatefulWidget {
    AnimationDemo01({Key key}): super(key: key);

    @override
```

```
_AnimationDemo01State createState() => _AnimationDemo01State();  
}  
  
class _AnimationDemo01State extends State<AnimationDemo01> with SingleTickerProviderStateMixin {  
  AnimationController controller;  
  Animation<double> animation;  
  
  Animation<Color> colorAnim;  
  Animation<double> sizeAnim;  
  Animation<double> rotationAnim;  
  
  @override  
  void initState() {  
    super.initState();  
  
    // 1. 创建AnimationController  
    controller = AnimationController(duration: Duration(seconds: 2), vsync: this);  
    // 2. 动画添加Curve效果  
    animation = CurvedAnimation(parent: controller, curve: Curves.easeIn);  
    // 3. 监听动画  
    animation.addListener(() {  
      setState(() {});  
    });  
  
    // 4. 设置值的变化  
    colorAnim = ColorTween(begin: Colors.blue, end: Colors.red).animate(controller);  
    sizeAnim = Tween(begin: 0.0, end: 200.0).animate(controller);  
    rotationAnim = Tween(begin: 0.0, end: 2*pi).animate(controller);  
  }  
  
  @override  
  Widget build(BuildContext context) {  
    return Center(  
      child: Opacity(  
        opacity: animation.value,  
        child: Transform(  
          alignment: Alignment.center,  
          transform: Matrix4.rotationZ(animation.value),  
          child: Container(  
            width: sizeAnim.value,  
            height: sizeAnim.value,  
            color: colorAnim.value,  
            alignment: Alignment.center,  
          ),  
        ),  
      ),  
    );  
  }  
  
  @override  
  void dispose() {  
    controller.dispose();  
    super.dispose();  
  }  
}
```

当然，我们可以使用Builder来对代码进行优化

```
@override
Widget build(BuildContext context) {
  return Center(
    child: AnimatedBuilder(
      animation: controller,
      builder: (ctx, child) {
        return Opacity(
          opacity: animation.value,
          child: Transform(
            alignment: Alignment.center,
            transform: Matrix4.rotationZ(rotationAnim.value),
            child: Container(
              width: sizeAnim.value,
              height: sizeAnim.value,
              color: colorAnim.value,
              alignment: Alignment.center,
            ),
          ),
        );
      },
    ),
  );
}
```

3.2. Hero动画

移动端开发会经常遇到类似这样的需求：

- 点击一个头像，显示头像的大图，并且从原来图像的Rect到大图的Rect
- 点击一个商品的图片，可以展示商品的大图，并且从原来图像的Rect到大图的Rect

这种跨页面共享的动画被称之为享元动画（Shared Element Transition）

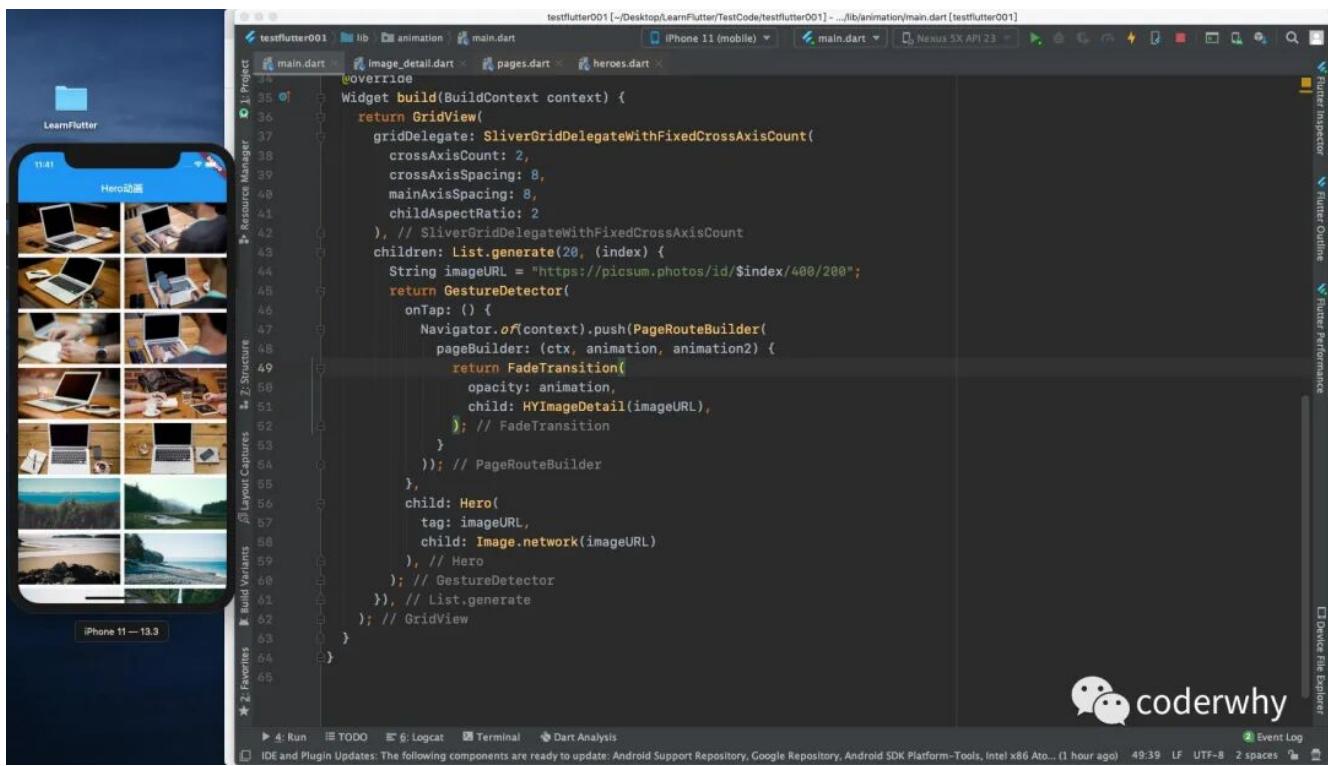
在Flutter中，有一个专门的Widget可以来实现这种动画效果：Hero

实现Hero动画，需要如下步骤：

- 1. 在第一个Page1中，定义一个起始的Hero Widget，被称之为source hero，并且绑定一个tag；
- 2. 在第二个Page2中，定义一个终点的Hero Widget，被称之为 destination hero，并且绑定相同的tag；
- 3. 可以通过Navigator来实现第一个页面Page1到第二个页面Page2的跳转过程；

Flutter会设置Tween来界定Hero从起点到终端的大小和位置，并且在图层上执行动画效果。

首页Page代码：



```
testflutter001 [~/Desktop/LearnFlutter/TestCode/testflutter001] - lib/main.dart [testflutter001]
override
Widget build(BuildContext context) {
  return GridView(
    gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
      crossAxisCount: 2,
      crossAxisSpacing: 8,
      mainAxisSpacing: 8,
      childAspectRatio: 2
    ), // SliverGridDelegateWithFixedCrossAxisCount
    children: List.generate(20, (index) {
      String imageURL = "https://picsum.photos/id/$index/400/200";
      return GestureDetector(
        onTap: () {
          Navigator.of(context).push(PageRouteBuilder(
            pageBuilder: (ctx, animation, animation2) {
              return FadeTransition(
                opacity: animation,
                child: HYImageDetail(imageURL),
              ); // FadeTransition
            }
          )); // PageRouteBuilder
        },
        child: Hero(
          tag: imageURL,
          child: Image.network(imageURL),
        ), // Hero
      ); // GestureDetector
    }), // List.generate
  ); // GridView
}

```

首页Page

```
import 'dart:math';

import 'package:flutter/material.dart';
import 'package:testflutter001/animation/image_detail.dart';

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

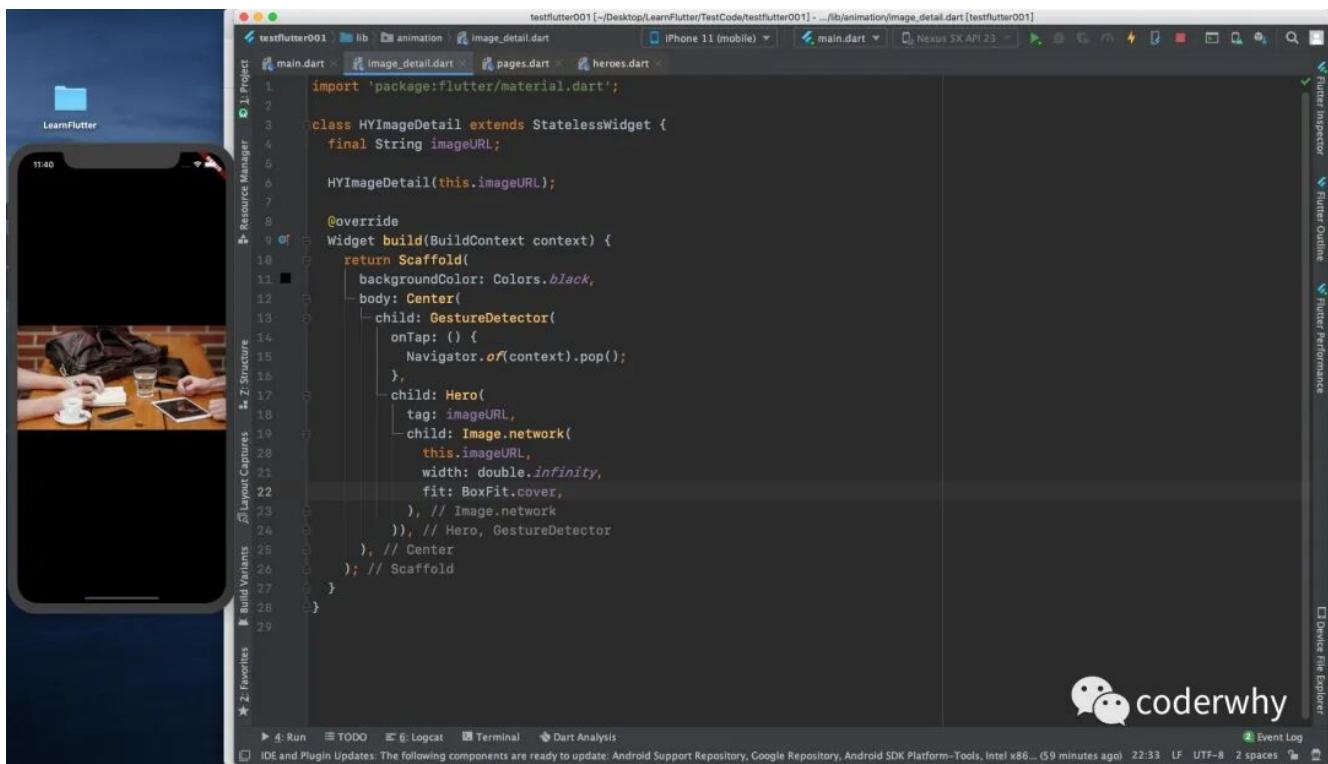
class MyApp extends StatelessWidget {
    // This widget is the root of your application.

    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            title: 'Flutter Demo',
            theme: ThemeData(
                primarySwatch: Colors.blue, splashColor: Colors.transparent),
            home: HYHomePage(),
        );
    }
}

class HYHomePage extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return Scaffold(
            appBar: AppBar(
                title: Text("Hero动画"),
            ),
            body: HYHomeContent(),
        );
    }
}

class HYHomeContent extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return GridView(
            gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
                crossAxisCount: 2,
                crossAxisSpacing: 8,
                mainAxisSpacing: 8,
                childAspectRatio: 2
            ),
            children: List.generate(20, (index) {
                String imageURL = "https://picsum.photos/id/$index/400/200";
                return GestureDetector(
                    onTap: () {
                        Navigator.of(context).push(PageRouteBuilder(
                            pageBuilder: (ctx, animation, animation2) {
                                return FadeTransition(
                                    opacity: animation,
                                    child: HYImageDetail(imageURL),
                                );
                            }
                        )));
                    },
                    child: Hero(
                        tag: imageURL,
                        child: Image.network(imageURL)
                    ),
                );
            }),
        );
    }
}
```

图片展示Page



图片展示Page

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

class HYImageDetail extends StatelessWidget {
    final String imageURL;

    HYImageDetail(this.imageURL);

    @override
    Widget build(BuildContext context) {
        return Scaffold(
            backgroundColor: Colors.black,
            body: Center(
                child: GestureDetector(
                    onTap: () {
                        Navigator.of(context).pop();
                    },
                    child: Hero(
                        tag: imageURL,
                        child: Image.network(
                            this.imageURL,
                            width: double.infinity,
                            fit: BoxFit.cover,
                        ), // Image.network
                    ), // Hero, GestureDetector
                ), // Center
            ); // Scaffold
    }
}
```

备注：所有内容首发于公众号，之后除了Flutter也会更新其他技术文章，TypeScript、React、Node、uniapp、mpvue、数据结构与算法等等，也会更新一些自己的学习心得等，欢迎大家关注



coderwhy

微信扫描二维码，关注我的公众号

 coderwhy

公众号