Animations

Animation<double>

Во Flutter объект Animation ничего не знает о том, что находится на экране. Анимация - это абстрактный класс, который понимает свое текущее значение и состояние (завершено или отклонено). Один из наиболее часто используемых типов анимации - это Animation <double>.

Объект Animation последовательно генерирует интерполированные числа между двумя значениями в течение определенного периода времени. Вывод объекта Animation может быть линейным, кривой, ступенчатой функцией или любым другим отображением, которое вы можете придумать. В зависимости от того, как управляется объект Animation, он может работать в обратном направлении или даже переключать направления посередине.

Анимации также могут интерполировать типы, отличные от double, например Animation <Color> или Animation <Size>.

Объект Animation имеет состояние. Его текущее значение всегда доступно в члене .value.

Объект Animation ничего не знает о функциях рендеринга или build ().

CurvedAnimation (extend Animation<double>)

CurvedAnimation определяет ход анимации как нелинейную кривую.

```
animation = CurvedAnimation(parent: controller, curve: Curves.easeIn);
```

Custom Curve

```
import 'dart:math';

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

AnimationController

AnimationController - это специальный объект Animation, который генерирует новое значение всякий раз, когда устройство готово к новому кадру.

По умолчанию AnimationController линейно производит числа от 0,0 до 1,0 в течение заданного времени.

Генерация чисел связана с обновлением экрана, поэтому обычно в секунду генерируется 60 чисел.

controller = AnimationController(duration: const Duration(seconds: 2), vsync: this);

При создании AnimationController вы передаете ему аргумент vsync. Наличие vsync предотвращает потребление ненужных ресурсов анимацией за кадром.

Tween

По умолчанию объект AnimationController находится в диапазоне от 0,0 до 1,0. Если вам нужен другой диапазон или другой тип данных, вы можете использовать Tween для настройки анимации для интерполяции в другой диапазон или другой тип данных. Например, следующий Tween изменяется от -200,0 до 0,0:

```
tween = Tween<double>(begin: -200, end: 0);
```

Animation notifications

Объект Animation может иметь Listeners и StatusListener, определенные с помощью addListener () и addStatusListener ().

Listener вызывается всякий раз, когда изменяется значение анимации. Наиболее распространенное поведение Listener - это вызов setState (), чтобы вызвать перестроение.

StatusListener вызывается, когда анимация начинается, заканчивается, движется вперед или движется назад, как определено AnimationStatus.

Example

```
void main() => runApp(LogoApp());

class LogoApp extends StatefulWidget {
  _LogoAppState createState() => _LogoAppState();
}

class _LogoAppState extends State<LogoApp> {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: Container(
        margin: EdgeInsets.symmetric(vertical: 10),
        height: 300,
        width: 300,
        child: FlutterLogo(),
      ),
    );
  }
}
```

```
class _LogoAppState extends State<LogoApp> with SingleTickerProviderStateMixin {
Animation<double> animation;
AnimationController controller;
@override
void initState() {
super.initState();
controller =AnimationController(duration: const Duration(seconds: 2), vsync: this);
animation = Tween<double>(begin: 0, end: 300).animate(controller)
        ..addListener(() {
                setState(() {
// The state that has changed here is the animation object’s value.
});
});
controller.forward();
}
```

```
class _LogoAppState extends State<LogoApp> {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: Container(
        margin: EdgeInsets.symmetric(vertical: 10),
        height: animation.value,
        width: animation.value,
        child: FlutterLogo(),
      ),
    );
  }
}

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

AnimatedBuilder

Constructors

AnimatedBuilder({Key? key, required Listenable animation, required TransitionBuilder builder, Widget? child})

Creates an animated builder. [...]

Properties

builder → TransitionBuilder

Called every time the animation changes value.

Example

```
AnimationController _controller = AnimationController(
  duration: const Duration(seconds: 10),
  vsync: this,
)..repeat();
```

```
@override
Widget build(BuildContext context) {
  return AnimatedBuilder(
    animation: _controller,
    child: Container(
      width: 200.0,
      height: 200.0,
      color: Colors.green,
      child: const Center(
        child: Text('Whee!'),
      ),
    ),
    builder: (BuildContext context, Widget? child) {
      return Transform.rotate(
        angle: _controller.value * 2.0 * math.pi,
        child: child,
      );
    },
  );
}
```

AnimatedOpacity

Constructors

AnimatedOpacity({Key? key, Widget? child, required double opacity, Curve curve: Curves.linear, required Duration duration, VoidCallback? onEnd, bool alwaysIncludeSemantics: false})

Creates a widget that animates its opacity implicitly. [...]

Properties

alwaysIncludeSemantics → bool

Whether the semantic information of the children is always included. [...]

child → Widget?

The widget below this widget in the tree. [...]

opacity → double

The target opacity. [...]

AnimatedPadding

Constructors

AnimatedPadding({Key? key, required EdgeInsetsGeometry padding, Widget? child, Curve curve: Curves.linear, required Duration duration, VoidCallback? onEnd})

Creates a widget that insets its child by a value that animates implicitly. [...]

Properties

child → Widget?

The widget below this widget in the tree. [...]

padding → EdgeInsetsGeometry

The amount of space by which to inset the child.

AnimatedPositioned

Constructors

AnimatedPositioned({Key? key, required Widget child, double? left, double? top, double? right, double? bottom, double? width, double? height, Curve curve: Curves.linear, required Duration duration, VoidCallback? onEnd})

Creates a widget that animates its position implicitly. [...]

AnimatedPositioned.fromRect({Key? key, required Widget child, required Rect rect, Curve curve: Curves.linear, required Duration duration, VoidCallback? onEnd})

Creates a widget that animates the rectangle it occupies implicitly. [...]

Properties

bottom → double?
The offset of the child's bottom edge from the bottom of the stack.

child → Widget
The widget below this widget in the tree. [...]

height → double?
The child's height. [...]

left → double?
The offset of the child's left edge from the left of the stack.

right → double?
The offset of the child's right edge from the right of the stack.

top → double?
The offset of the child's top edge from the top of the stack.

width → double?
The child's width. [...]

Flare-Flutter

```
dependencies: flare_flutter: ^2.0.6

import 'package:flare_flutter/flare_actor.dart';
class MyHomePage extends StatefulWidget {
  @override
  _MyHomePageState createState() => new _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  @override
  Widget build(BuildContext context) {
    return new FlareActor("assets/Filip.flr", alignment:Alignment.center, fit:BoxFit.contain, animation:"idle");
  }
}
```