本节目标:

```
[1]. 认识动画器 [曲线速率] 的作用。
[2]. 绘制出 Flutter 内置的所有曲线效果常量。
[3]. 了解动画器的常用方法。
[4]. 了解动画器的状态，及状态变化监听。
```

一、动画器速率曲线

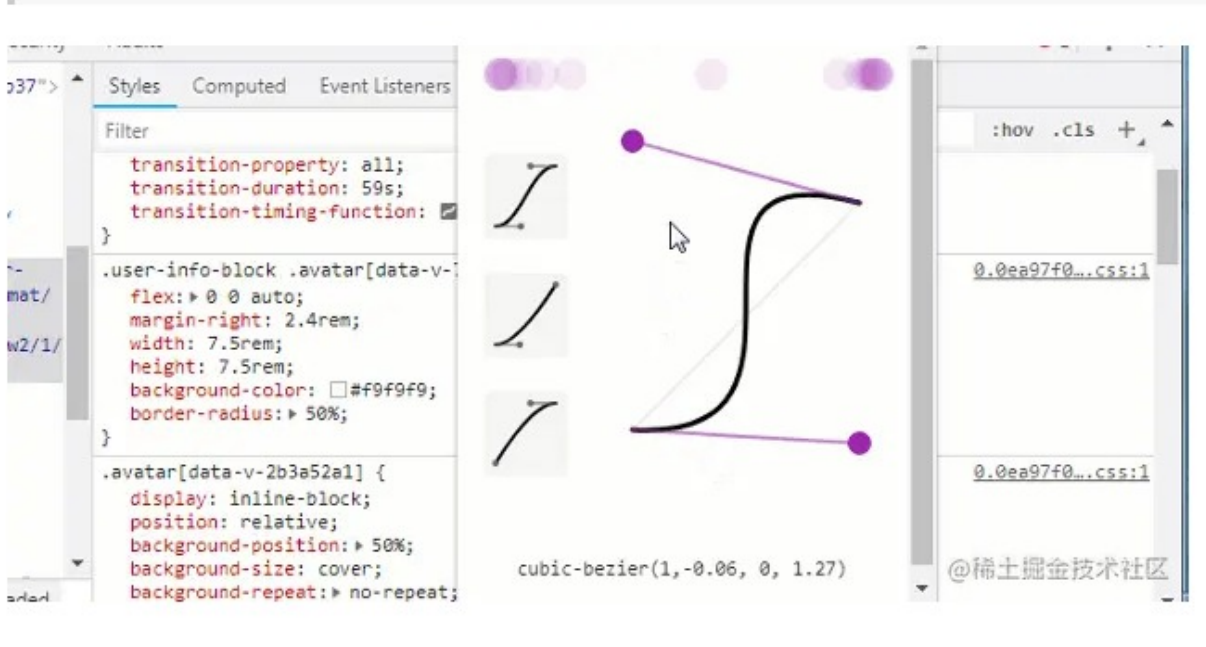
1. 认识曲线

> 动画曲线描述了 动画运动过程中的速度变化速率 ，比如开始运动很快，然后慢慢变慢，这样可以实现丰富的运动视觉效果。
> 如下，在 Chrome 浏览器 中如果有变换附加运动曲线的话，可以点击查看曲线的详情。

```
.lazy[data-v-6dcfedb8]:not(.immediate):before {
  transition: ▾ opacity .2s;
    transition-property: opacity;
    transition-duration: 0.2s;
    transition-timing-function: cubic-bezier(0.25, 0.1, 0.05, 0.64);
    transition-delay: 0s;
}
```

```
.lazy[data-v-6dcfedb8]:not(.immediate):before {
  transition: ▾ opacity .2s;
    transition-property: opacity;
    transition-duration: 0.2s;
    transition-timing-function: ease;
    transition-delay: 0s;
}
```

> 动画曲线的核心是 四个数字 ，它的本身是一个 贝塞尔曲线 。起止点固定，也就是说动画曲线在形式上的表现就是两个控制点。Flutter 中使用 Curve 类 来表述。



```
transition-property: all;
transition-duration: 59s;
transition-timing-function:
}
.user-info-block .avatar[data-v-
  flex: ▸ 0 0 auto;
  margin-right: 2.4rem;
  width: 7.5rem;
  height: 7.5rem;
  background-color: #f9f9f9;
  border-radius: ▸ 50%;
}
.avatar[data-v-2b3a52a1] {
  display: inline-block;
  position: relative;
  background-position: ▸ 50%;
  background-size: cover;
  background-repeat: ▸ no-repeat;
```

---

2. 认识的使用

> Curve 是一个 抽象类 ，它有很多实现子类。最通用的是 Cubic ，传入四个值。下面来看一下，如何使用 CurveTween 让动画的变化速率具有曲线效果。

```dart
Curve --- 抽象类
        |--- Cubic 四值三阶贝塞尔曲线
        |--- FlippedCurve  翻转曲线
        |--- SawTooth 锯齿形曲线
        |--- Threshold 0~阈值 曲线
        |--- ...
```

> 案例中实现如下效果：红色小球在圈上运动，绿色小球向下运动。通过改变动画的速率曲线，查看效果。 可以发现，通过添加曲线可以让运动在首尾慢，中间快。

---

3. 实现测试组件

> 为了更好测试动画曲线效果，这里做了一个 CurveBox 组件，可以传入曲线，以便更好复用。

```dart
---->[p11_anim/s01_curve_diy/curve_box.dart]----
class CurveBox extends StatefulWidget {
  final Color color;
  final Curve curve;

  CurveBox({Key? key, this.color = Colors.lightBlue, this.curve = Curves.linear})
      : super(key: key);

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

class _CurveBoxState extends State<CurveBox>
    with SingleTickerProviderStateMixin {
  late AnimationController _controller;
  late Animation<double> _angleAnimation;

  @override
  void initState() {
    super.initState();
    _controller = AnimationController(
      duration: const Duration(seconds: 3),
      vsync: this,
    );
    _angleAnimation = CurveTween(curve: widget.curve).animate(_controller);
    _controller.repeat();
  }

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

  @override
  Widget build(BuildContext context) {
    return CustomPaint(
      size: Size(100, 100),
      painter: CurveBoxPainter(_controller, _angleAnimation), // 背景
    );
  }
}
```

---

4. 实现绘制代码

> 绘制的逻辑比较简单，值得注意的是 在绘制时 如何将 动画器的值 和 画布的变化 结合起来，达到小球旋转和移动的效果。可见下面 _drawRedCircle 和 _drawGreenCircle 的处理。

```dart
class CurveBoxPainter extends CustomPainter {
  final Animation<double> repaint;
  Animation<double> angleAnimation;
  Paint _paint = Paint();

  CurveBoxPainter(this.repaint, this.angleAnimation) : super(repaint: repaint);

  @override
  void paint(Canvas canvas, Size size) {
    canvas.clipRect(Offset.zero & size);
    canvas.translate(size.width / 2, size.height / 2);
    _drawRing(canvas, size);
    _drawRedCircle(canvas, size);
    _drawGreenCircle(canvas, size);
  }

  //绘制环
  void _drawRing(Canvas canvas, Size size) {
    final double strokeWidth = 5;
    _paint
      ..color = Colors.blue
      ..style = PaintingStyle.stroke
      ..strokeWidth = strokeWidth;
    canvas.drawCircle(Offset.zero, size.width / 2 - strokeWidth, _paint);
  }

  // 绘制红球
  void _drawRedCircle(Canvas canvas, Size size) {
    canvas.save();
    canvas.rotate(angleAnimation.value * 2 * pi);
    _paint
      ..color = Colors.red
      ..style = PaintingStyle.fill;
    canvas.drawCircle(
        Offset.zero.translate(0, -(size.width / 2 - 5)), 5, _paint);
    canvas.restore();
  }

  // 绘制绿球
  void _drawGreenCircle(Canvas canvas, Size size) {
    canvas.save();
    canvas.translate(0,angleAnimation.value * (size.height-10));
    _paint
      ..color = Colors.green
      ..style = PaintingStyle.fill;
    canvas.drawCircle(
        Offset.zero.translate(0, -(size.width / 2 - 5)), 5, _paint);
    canvas.restore();
  }

  @override
  bool shouldRepaint(covariant CurveBoxPainter oldDelegate) =>
      oldDelegate.repaint != repaint;
}
```

5.Flutter内置的曲线效果

> 当我们将曲线效果颜色 封装 成一个组件后，就可以很轻松的实现下面的效果，这就是 Flutter 的魅力。 一共 41 个曲线效果，大家可以结合动图，自己看看。

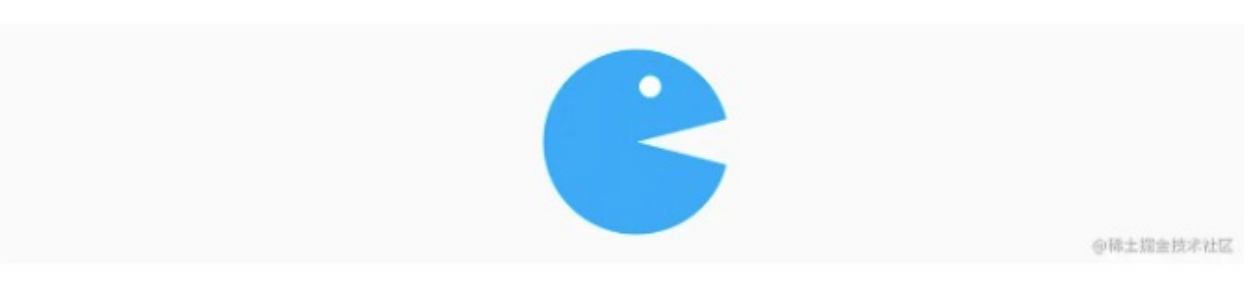
```dart
---->[p11_anim/s02_curve_show/main.dart]----
Wrap(
  runSpacing: 10,
  children: curvesMap.keys.map((e) => Column(
    mainAxisSize: MainAxisSize.min,
    children: [
      CurveBox(curve: curvesMap[e]!,),
      SizedBox(height: 3,),
      Text(e,style: TextStyle(fontSize: 10),)
    ],
  )).toList(),
),
```

二、动画器的方法

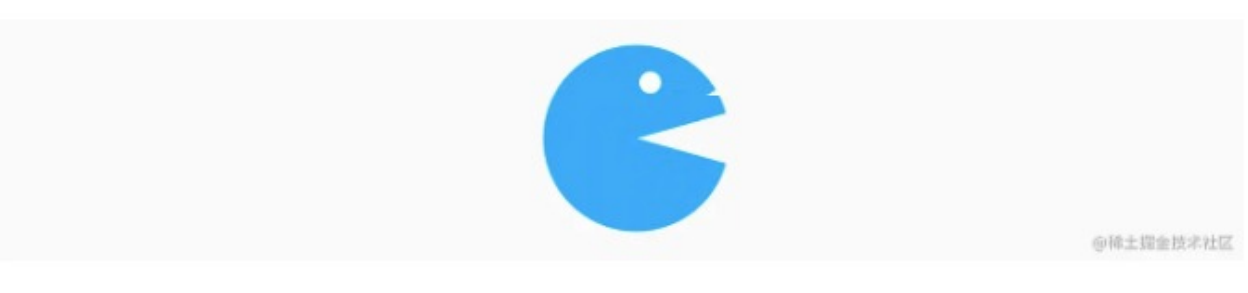
1. forward 和 reverse 方法

```dart
|--- forward({ double from })  执行运动直到上界，可指定初始值。(执行完就会停止)
|--- reverse({ double from })  执行运动直到下界，可指定初始值。(执行完就会停止)
```

> `_controller.forward();` 效果如下，角度从 0 逐渐变化到 40 ，然后停止动画。



> `_controller.reverse(from: 40);` 效果如下，角度从 40 逐渐变化到 0 ，然后停止动画。 注意： 动画器的值本身就是在下界，如果一开始就使用reverse运动到下届，是不会动的。

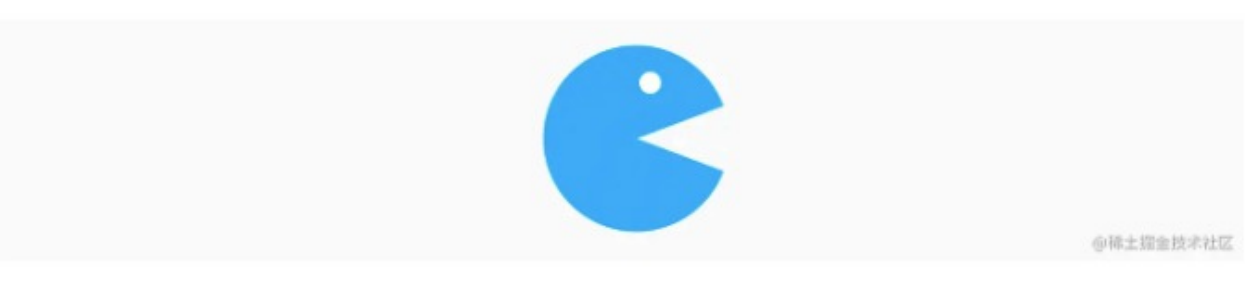


---

2. repeat 重复执行

```dart
repeat
        |--- double? min, // 下界
        |--- double? max, // 上界
        |--- bool reverse = false, // 是否反转
        |--- Duration? period // 周期
```

> repeat 方法默认是 不反转 的， 也就是 `0->1` 、`0->1` 、`0->1` 、`0->1...` 这样一直运动。

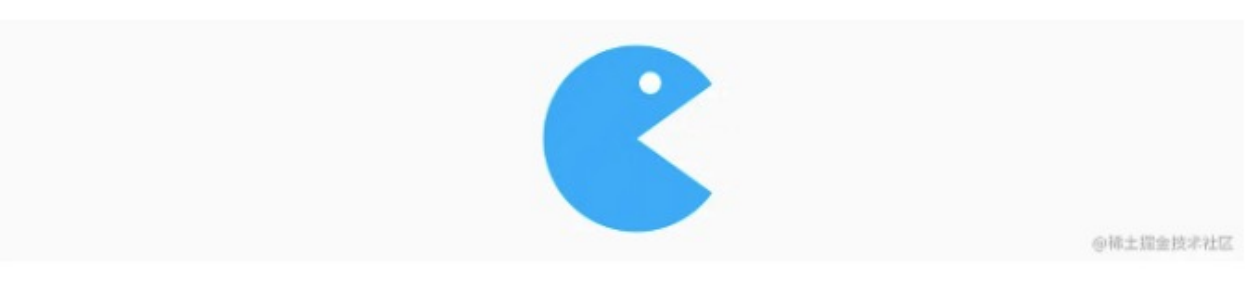


> repeat 方法设置 反转 `true` 时， 也就是 `0->1` 、`1->0` 、`0->1` 、`1->0 ...` 这样一直运动。

---

3. fling 方法

> fling 也就是 猛冲 ，可以给一个速度，默认为 1 。



---

4. stop 和 reset

> stop 方法是让 `_ticker` 停止，这样动画也就停止了。可以传入一个 bool 值，用于是否取消。 reset 方法也很简单，就是将当前值置为 下界 。

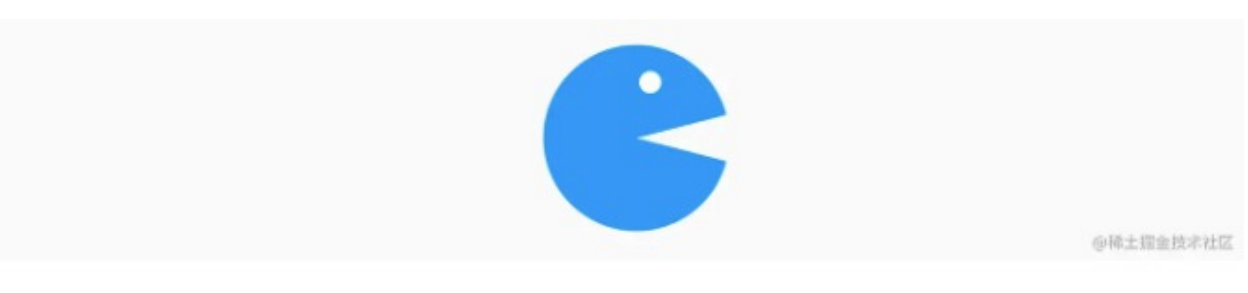
```dart
void stop({ bool canceled = true }) {
  _simulation = null;
  _lastElapsedDuration = null;
  _ticker!.stop(canceled: canceled);
}

void reset() {
  value = lowerBound;
}
```

三、动画器的状态

> 动画器的状态定义在 AnimationStatus 枚举中，可以通过添加监听器的来获取动画器状态。 `void addStatusListener(AnimationStatusListener listener)`

```dart
enum AnimationStatus {
  /// The animation is stopped at the beginning
  dismissed, //在正在开始时停止了
  /// The animation is running from beginning to end
  forward,//运动中
  /// The animation is running backwards, from end to beginning
  reverse,// 跑到终点，再跑回来的时候
  /// The animation is stopped at the end
  completed,//跑到终点时
}
```

> 下面我们就通过一个小例子来测试一下。比如在不同状态时显示不同的颜色值。 这里有个很重要的要点，就是 画板如何接受多个可监听属性 。



> CustomPainter 中建议使用 Listenable 对象来控制画布的刷新。这样是最高效的方式。 所以不要再用 setState 来刷新画板了。那么问题来了，如何使用多个可监听的属性呢？ 我通过查看 Flutter 中某些组件的绘制逻辑，发现了解决方案，使用 Listenable.merge 可以合并多个可监听对象。

```dart
---->[p11_anim/s04_animate_status/pic_man.dart]----
class _PicManState extends State<PicMan> with SingleTickerProviderStateMixin {
  late AnimationController _controller;
  final ValueNotifier<Color> _color = ValueNotifier<Color>(Colors.blue);

  @override
  void initState() {
    super.initState();
    _controller = AnimationController(
      lowerBound: 10,
      upperBound: 40,
      duration: const Duration(seconds: 1),
      vsync: this,
    );
    _controller.addStatusListener(_statusListen);
    _controller.forward();
    // _controller.repeat(reverse: true);
  }

  @override
  Widget build(BuildContext context) {
    return CustomPaint(
      size: Size(100, 100),
      painter: PicManPainter(
          color: _color,
          angle: _controller,
          repaint: Listenable.merge([_controller, _color])), // 背景
    );
  }

  void _statusListen(AnimationStatus status) {
      switch(status){
        case AnimationStatus.dismissed:
          _color.value = Colors.black;
          break;
        case AnimationStatus.forward:
          _color.value = Colors.blue;
          break;
        case AnimationStatus.reverse:
          _color.value = Colors.red;
          break;
        case AnimationStatus.completed:
          _color.value = Colors.green;
          break;
      }
  }
}
```

> 在画板中使用合在一起的可监听对象为 repaint 赋值。

```dart
class PicManPainter extends CustomPainter {
  final Animation<double> angle;
  final Listenable repaint;
  final ValueNotifier<Color> color;

  PicManPainter({required this.color,required this.angle, required this.repaint})
      : super(repaint: repaint);

  Paint _paint = Paint();

  @override
  void paint(Canvas canvas, Size size) {
    canvas.clipRect(Offset.zero & size); //剪切画布
    final double radius = size.width / 2;
    canvas.translate(radius, size.height / 2);
    _drawHead(canvas, size);
    _drawEye(canvas, radius);
  }

  //绘制头
  void _drawHead(Canvas canvas, Size size) {
    var rect = Rect.fromCenter(
        center: Offset(0, 0), height: size.width, width: size.height);
    var a = angle.value / 180 * pi;
    canvas.drawArc(
        rect, a, 2 * pi - a.abs() * 2, true, _paint..color = color.value);
  }

  //绘制眼睛
  void _drawEye(Canvas canvas, double radius) {
    canvas.drawCircle(Offset(radius * 0.15, -radius * 0.6), radius * 0.12,
        _paint..color = Colors.white);
  }

  @override
  bool shouldRepaint(covariant PicManPainter oldDelegate) =>
      oldDelegate.repaint != repaint;
}
```

> 到此为止，动画的使用就结束了，在之后的篇章中也会经常使用到动画。接下来我们将看 手势操作 在画版中的使用，这点也尤为重要。

留言

输入评论（Enter换行，⌘ + Enter发送）

发表评论

全部评论（10）

风二中 LV.1 JY.4 | Android&Flutter | 1月前
一、3. 实现测试组件，painter: PicManPainter(_controller, _angleAnimation)，PicManPainter应该是CurveBoxPainter吧？😁
点赞 1

张风捷特烈（作者） | 1月前
是的，已更正😁
点赞 回复

衿璃 JY.4 | flutter | 3月前
公司被封了，oh!😭不学了！回家睡觉
1 1

衿璃 | 3月前
😩天选的打工人
点赞 回复

混子中的混子 JY.3 | 前端混子 | 8月前
点解 🫰 🫰 🫰
点赞 回复

老李code LV.4 JY.5 | Android、Flutter开发 | 11月前
打卡💪
点赞 回复

ZeroFlutter LV.4 JY.4 | Flutter 高级工程师 @ Fl... | 1年前
前面的章节还考虑多个属性如何监听，这篇就讲到了 Listenable.merge ，赞
1 回复

旺仔139 JY.1 | 2年前
动画系列搞不懂的话可以去看看 b站王叔不秃的视频 讲的挺清晰
3 回复

煮一壶月光下酒 JY.1 | 酒徒 | 2年前
😍
点赞 回复

Tick(杜鹏) LV.1 JY.3 | 2年前
👏
点赞 回复