本章目录

自定义Widget方法简介

当Flutter提供的现有Widget无法满足我们的需求，或者我们为了共享代码需要封装一些通用Widget，这时我们就需要自定义Widget。在Flutter中自定义Widget有三种方式：通过组合其它Widget、自绘和实现RenderObject，本节我们先分别介绍一下这三种方式的特点，后面章节中则详细介绍它们的细节。

组合其它Widget

这种方式是通过拼装其它低级别的Widget来组合成一个高级别的Widget，例如我们之前介绍的Container就是一个组合Widget，它是由DecoratedBox、ConstrainedBox、Transform、Padding、Align等组成。

在Flutter中，组合的思想非常重要，Flutter提供了非常多的基础Widget，而我们的界面开发都是按照需要组合这些Widget来实现各种不同的布局。

自绘

如果遇到无法通过系统提供的现有Widget实现的UI时，如我们需要一个渐变圆形进度条，而Flutter提供的CircularProgressIndicator并不支持在显示精确进度时对进度条应用渐变色（其`valueColor`属性只支持执行旋转动画时变化Indicator的颜色），这时最好的方法就是通过自定义Widget绘制逻辑来画出我们期望的外观。Flutter中提供了CustomPaint和Canvas供我们自绘UI外观。

实现RenderObject

Flutter提供的任何具有UI外观的Widget，如文本Text、Image都是通过相应的RenderObject渲染出来的，如Text是由RenderParagraph渲染，而Image是由RenderImage渲染。RenderObject是一个抽象类，它定义了一个抽象方法`paint(...)`：

```
void paint(PaintingContext context, Offset offset)
```

PaintingContext代表Widget的绘制上下文，通过`PaintingContext.canvas`可以获得Canvas，绘制逻辑主要是通过Canvas API来实现。子类需要实现此方法以实现自身的绘制逻辑，如RenderParagraph需要实现文本绘制逻辑，而RenderImage需要实现图片绘制逻辑。

可以发现，RenderObject中最终也是通过Canvas来绘制的，那么通过实现RenderObject的方式和上面介绍的通过CustomPaint和Canvas自绘的方式有什么区别？其实答案很简单，CustomPaint只是为了方便开发者封装的一个代理类，它直接继承自SingleChildRenderObjectWidget，通过RenderCustomPaint的paint方法将Canvas和画笔Painter(需要开发者实现，后面章节介绍)连接起来实现了最终的绘制（绘制逻辑在Painter中）。

总结

组合是自定义组件最简单的方法，在任何需要自定义的场景下，都应该优先考虑是否能够通过组合来实现。而自绘和通过实现RenderObject的方法本质上是一样的，都需要开发者调用Canvas API手动去绘制UI，缺点时必须了解Canvas API，并且得自己去实现绘制逻辑，而优点是强大灵活，理论上可以实现任何外观的UI。

在本章接下来的小节中，我们将通过一些实例来详细介绍自定义UI的过程，由于后两种方法本质是相同的，后续我们只介绍CustomPaint和Canvas的方式，读者如果对自定义RenderObject的方法好奇，可以查看RenderParagraph或RenderImage源码。

通过组合现有Widget实现自定义Widget

在Flutter中页面UI通常都是由一些低阶别的Widget组合而成，当我们需要封装一些通用Widget时，应该首先考虑是否可以通过组合其它Widget来实现，如果可以则应优先使用组合，因为直接通过现有Widget拼装会非常方便、简单、高效。

示例：自定义渐变按钮

Flutter Widget库中的按钮默认不支持渐变背景，为了实现渐变背景按钮，我们自定义一个GradientButton Widget。我们先来看看效果：

我们看看GradientButton实现：

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

class GradientButton extends StatelessWidget {
  GradientButton({
    this.colors,
    this.width,
    this.height,
    this.onTap,
    @required this.child,
  });

  // 渐变色数组
  final List<Color> colors;

  // 按钮宽高
  final double width;
  final double height;

  final Widget child;

  //点击回调
  final GestureTapCallback onTap;

  @override
  Widget build(BuildContext context) {
    ThemeData theme = Theme.of(context);

    //确保colors数组不空
    List<Color> _colors = colors ??
        [theme.primaryColor, theme.primaryColorDark ??
theme.primaryColor];
```

```
    return DecoratedBox(
      decoration: BoxDecoration(
        gradient: LinearGradient(colors: _colors),
      ),
      child: Material(
        type: MaterialType.transparency,
        child: InkWell(
          splashColor: colors.last,
          highlightColor: Colors.transparent,
          onTap: onTap,
          child: ConstrainedBox(
            constraints: BoxConstraints.tightFor(height: height,
width: width),
            child: Center(
              child: Padding(
                padding: const EdgeInsets.all(8.0),
                child: DefaultTextStyle(
                    style: TextStyle(fontWeight: FontWeight.bold),
                    child: child),
              ),
            ),
          ),
        ),
      ),
    );
  }
}
```

可以看到GradientButton是由Padding、Center、InkWell等Widget组合而成。当然上面的代码只是一个示例，作为一个按钮它还并不完整，比如没有禁用状态、不能定义圆角等，读者可以根据实际需要来完善。

使用GradientButton

```
class GradientButtonRoute extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
      child: Column(
        children: <Widget>[
          GradientButton(
            colors: [Colors.orange,Colors.red],
            height: 50.0,
            child: Text("Submit"),
            onTap:onTap ,
          ),
          GradientButton(
            height: 50.0,
            colors: [Colors.lightGreen, Colors.green[700]],
            child: Text("Submit"),
            onTap: onTap,
          ),
          GradientButton(
            height: 50.0,
            colors: [Colors.lightBlue[300], Colors.blueAccent],
            child: Text("Submit"),
            onTap: onTap,
          ),
        ],
      ),
    );
  }

  onTap(){
    print("button click");
  }
}
```

总结

通过组合的方式定义Widget和我们之前写界面并无差异，不过在抽离出单独的Widget时我们要考虑代码规范性，如必要参数要用`@required`标注，对于可选参数在特定场景需要判空或设置默认值等。这是由于使用者大多时候可能不了解Widget的内部细节，所以为了保证代码健壮性，我们需要在用户错误地使用Widget时能够兼容或报错提示（使用assert断言函数）。

实例：TurnBox

我们之前已经介绍过RotatedBox，但是它有两个缺点：一是只能将其子节点以90度的倍数旋转，二是当旋转的角度发生变化时，旋转角度更新过程没有动画。

本节我们将实现一个TurnBox，它可以以任意角度来旋转其子节点，并且在角度发生变化时可以执行一个动画过渡到新状态，同时，我们可以手动指定动画速度。

TurnBox的完整代码如下：

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

class TurnBox extends StatefulWidget {
  const TurnBox({
    Key key,
    this.turns = .0, //旋转的“圈”数,一圈为360度，如0.25圈即90度
    this.speed = 200, //过渡动画执行的总时长
    this.child
  }) :super(key: key);

  final double turns;
  final int speed;
  final Widget child;

  @override
  _TurnBoxState createState() => new _TurnBoxState();
}

class _TurnBoxState extends State<TurnBox>
    with SingleTickerProviderStateMixin {
  AnimationController _controller;

  @override
  void initState() {
    super.initState();
    _controller = new AnimationController(
        vsync: this,
        lowerBound: -double.infinity,
        upperBound: double.infinity
    );
    _controller.value = widget.turns;
  }

  @override
```

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

  @override
  Widget build(BuildContext context) {
    return RotationTransition(
      turns: _controller,
      child: widget.child,
    );
  }

  @override
  void didUpdateWidget(TurnBox oldWidget) {
    super.didUpdateWidget(oldWidget);
    //旋转角度发生变化时执行过渡动画
    if (oldWidget.turns != widget.turns) {
      _controller.animateTo(
        widget.turns,
        duration: Duration(milliseconds: widget.speed??200),
        curve: Curves.easeOut,
      );
    }
  }
}
```

代码比较简单，我们主要是通过包装(组合)RotationTransition来实现的。

下面我们测试一下TurnBox的功能，测试代码如下：

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

class TurnBoxRoute extends StatefulWidget {
  @override
  _TurnBoxRouteState createState() => new _TurnBoxRouteState();
}

class _TurnBoxRouteState extends State<TurnBoxRoute> {
  double _turns = .0;

  @override
  Widget build(BuildContext context) {

    return Center(
```

```
      child: Column(
        children: <Widget>[
          TurnBox(
            turns: _turns,
            speed: 500,
            child: Icon(Icons.refresh, size: 50,),
          ),
          TurnBox(
            turns: _turns,
            speed: 1000,
            child: Icon(Icons.refresh, size: 150.0,),
          ),
          RaisedButton(
            child: Text("顺时针旋转1/5圈"),
            onPressed: () {
              setState(() {
                _turns += .2;
              });
            },
          ),
          RaisedButton(
            child: Text("逆时针旋转1/5圈"),
            onPressed: () {
              setState(() {
                _turns -= .2;
              });
            },
          )
        ],
      ),
    );
  }
}
```

测试代码运行后效果如下图：

当我们点击旋转按钮时，两个图标的旋转都会旋转1/5圈，但旋转的速度是不同的，读者可以自己运行一下示例看看效果。

CustomPaint与Canvas

对于一些复杂或不规则的UI，我们可能无法使用现有Widget组合的方式来实现，比如我们需要一个正六边形、一个渐变的圆形进度条、一个棋盘等，当然有时候我们可以使用图片来实现，但在一些需要动态交互的场景静态图片是实现不了的，比如要实现一个手写输入面板。这时，我们就需要来自己绘制UI外观。

几乎所有的UI系统都会提供一个自绘UI的接口，这个接口通常会提供一块2D画布Canvas，Canvas内部封装了一些基本绘制的API，开发者可以通过Canvas绘制各种自定义图形。在Flutter中，提供了一个CustomPaint Widget，它可以结合一个画笔CustomPainter来实现绘制自定义图形。

CustomPaint

我们看看CustomPaint构造函数：

```
const CustomPaint({
  Key key,
  this.painter,
  this.foregroundPainter,
  this.size = Size.zero,
  this.isComplex = false,
  this.willChange = false,
  Widget child, //子节点，可以为空
})
```

- painter: 背景画笔，会显示在子节点后面;
- foregroundPainter: 前景画笔，会显示在子节点前面
- size：当child为null时，代表默认绘制区域大小，如果有child则忽略此参数，画布尺寸则为child尺寸。如果有child但是想指定画布为特定大小，可以使用SizeBox包裹CustomPaint实现。
- isComplex：是否复杂的绘制，如果是，Flutter会应用一些缓存策略来减少重复渲染的开销。
- willChange：和isComplex配合使用，当启用缓存时，该属性代表在下一帧中绘制是否会改变。

可以看到，绘制时我们需要提供前景或背景画笔，两者也可以同时提供。我们的画笔需要继承CustomPainter类，我们在画笔类中实现真正的绘制逻辑。

注意

如果CustomPaint有子节点，为了避免子节点不必要的重绘，通常情况下都会将子节点包裹在RepaintBoundary Widget中来隔离子节点和CustomPaint本身的绘制边界，如：

```
CustomPaint(
  size: Size(300, 300), //指定画布大小
  painter: MyPainter(),
  child: RepaintBoundary(child:...)),
)
```

CustomPainter

CustomPainter中提定义了一个虚函数`paint`：

```
void paint(Canvas canvas, Size size);
```

`paint`有两个参数:

- Canvas：一个画布，包括各种绘制方法，我们列出一下常用的方法：

API名称	功能
drawLine	画线
drawPoint	画点
drawPath	画路径
drawImage	画图像
drawRect	画矩形
drawCircle	画圆
drawOval	画椭圆
drawArc	画圆弧

- Size：当前绘制区域大小。

画笔Paint

现在画布有了，我们最后还缺一个画笔，Flutter提供了Paint类来实现画笔。在Paint中，我们可以配置画笔的各种属性如粗细、颜色、样式等。如：

```
var paint = Paint() //创建一个画笔并配置其属性
  ..isAntiAlias = true //是否抗锯齿
  ..style = PaintingStyle.fill //画笔样式：填充
  ..color=Color(0x77cdb175);//画笔颜色
```

更多的配置属性读者可以参考Paint类定义。

示例：五子棋/盘

下面我们通过一个五子棋游戏中棋盘和棋子的绘制来演示自绘UI的过程，首先我们看一下我们的目标结果：

代码：

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

class CustomPaintRoute extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: CustomPaint(
        size: Size(300, 300), //指定画布大小
        painter: MyPainter(),
      ),
    );
  }
}

class MyPainter extends CustomPainter {
  @override
  void paint(Canvas canvas, Size size) {
    double eWidth = size.width / 15;
    double eHeight = size.height / 15;
```

```
    //画棋盘背景
    var paint = Paint()
      ..isAntiAlias = true
      ..style = PaintingStyle.fill //填充
      ..color = Color(0x77cdb175); //背景为纸黄色
    canvas.drawRect(Offset.zero & size, paint);

    //画棋盘网格
    paint
      ..style = PaintingStyle.stroke //线
      ..color = Colors.black87
      ..strokeWidth = 1.0;

    for (int i = 0; i <= 15; ++i) {
      double dy = eHeight * i;
      canvas.drawLine(Offset(0, dy), Offset(size.width, dy),
paint);
    }

    for (int i = 0; i <= 15; ++i) {
      double dx = eWidth * i;
      canvas.drawLine(Offset(dx, 0), Offset(dx, size.height),
paint);
    }

    //画一个黑子
    paint
      ..style = PaintingStyle.fill
      ..color = Colors.black;
    canvas.drawCircle(
      Offset(size.width / 2 - eWidth / 2, size.height / 2 - eHeight
/ 2),
      min(eWidth / 2, eHeight / 2) - 2,
      paint,
    );

    //画一个白子
    paint.color = Colors.white;
    canvas.drawCircle(
      Offset(size.width / 2 + eWidth / 2, size.height / 2 - eHeight
/ 2),
      min(eWidth / 2, eHeight / 2) - 2,
      paint,
    );
  }

  //在实际场景中正确利用此回调可以避免重绘开销，本示例我们简单的返回true
```

```
  @override
  bool shouldRepaint(CustomPainter oldDelegate) => true;
}
```

性能

绘制是比较昂贵的操作，所以我们在实现自绘控件时应该考虑到性能开销，下面是两条关于性能优化的建议：

- 尽可能的利用好`shouldRepaint`返回值；在UI树重新build时，控件在绘制前都会先调用该方法以确定是否有必要重绘；加入我们绘制的UI不依赖外部状态，那么就应该始终返回false，因为外部状态改变导致重新build时不会影响我们的UI外观；如果绘制依赖外部状态，那么我们就应该在shouldRepaint中判断依赖的状态是否改变，如果已改变则应返回`true`来重绘，反之则应返回`false`不需要重绘。

- 绘制尽可能多的分层；在上面五子棋的示例中，我们将棋盘和棋子的绘制放在了一起，这样会有一个问题：由于棋盘始终是不变的，用户每次落子时变的只是棋子，但是如果按照上面的代码来实现，每次绘制棋子时都要重新绘制一次棋盘，这是没必要的。优化的方法就是将棋盘单独抽为一个Widget，并设置其`shouldRepaint`回调值为false，然后将棋盘Widget作为背景。然后将棋子的绘制放到另一个Widget中，这样落子时只需要绘制棋子。

总结

自绘控件非常强大，理论上可以实现任何2D图形外观，实际上Flutter提供的所有Widget最终都是调用Canvas绘制出来的，只不过绘制的逻辑被封装起来了，读者有兴趣可以查看具有外观样式的Widget的源码，找到其对应的RenderObject对象，如Text Widget最终会通过RenderParagraph对象来通过Canvas实现文本绘制逻辑。

下一节我们再通过实现一个自绘的圆形渐变进度条的实例来帮助读者加深印象。

实例：圆形渐变进度条(自绘)

本节我们实现一个圆形渐变进度条，它支持：

1. 支持多种渐变色。
2. 任意弧度；整个进度可以不是整圆。
3. 可以自定义粗细、两端是否圆角等样式。

可以发现要实现这样的一个进度条是无法通过现有组件组合而成的，所以我们通过自绘方式实现。

实现代码如下：

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

class GradientCircularProgressIndicator extends StatelessWidget {
  GradientCircularProgressIndicator({
    this.stokeWidth = 2.0,
    @required this.radius,
    @required this.colors,
    this.stops,
    this.strokeCapRound = false,
    this.backgroundColor = const Color(0xFFEEEEEE),
    this.totalAngle = 2 * pi,
    this.value
  });

  ///粗细
  final double stokeWidth;

  /// 圆的半径
  final double radius;

  ///两端是否为圆角
  final bool strokeCapRound;

  /// 当前进度，取值范围 [0.0-1.0]
  final double value;

  /// 进度条背景色
  final Color backgroundColor;

  /// 进度条的总弧度，2*PI为整圆，小于2*PI则不是整圆
  final double totalAngle;

  /// 渐变色数组
  final List<Color> colors;
```

```dart
  /// 渐变色的终止点，对应colors属性
  final List<double> stops;

  @override
  Widget build(BuildContext context) {
    double _offset = .0;
    // 如果两端为圆角，则需要对起始位置进行调整，否则圆角部分会偏离起始位置
    // 下面调整的角度的计算公式是通过数学几何知识得出，读者有兴趣可以研究一下为什么是这样
    if (strokeCapRound) {
      _offset = asin(stokeWidth / (radius * 2 - stokeWidth));
    }
    var _colors = colors;
    if (_colors == null) {
      Color color = Theme
          .of(context)
          .accentColor;
      _colors = [color, color];
    }
    return Transform.rotate(
      angle: -pi / 2.0 - _offset,
      child: CustomPaint(
          size: Size.fromRadius(radius),
          painter: _GradientCircularProgressPainter(
            stokeWidth: stokeWidth,
            strokeCapRound: strokeCapRound,
            backgroundColor: backgroundColor,
            value: value,
            total: totalAngle,
            radius: radius,
            colors: _colors,
          )
      ),
    );
  }
}

//实现画笔
class _GradientCircularProgressPainter extends CustomPainter {
  _GradientCircularProgressPainter({
    this.stokeWidth: 10.0,
    this.strokeCapRound: false,
    this.backgroundColor = const Color(0xFFEEEEEE),
    this.radius,
    this.total = 2 * pi,
    @required this.colors,
    this.stops,
    this.value
```

```
  });

  final double stokeWidth;
  final bool strokeCapRound;
  final double value;
  final Color backgroundColor;
  final List<Color> colors;
  final double total;
  final double radius;
  final List<double> stops;

  @override
  void paint(Canvas canvas, Size size) {
    if (radius != null) {
      size = Size.fromRadius(radius);
    }
    double _offset = stokeWidth / 2.0;
    double _value = (value ?? .0);
    _value = _value.clamp(.0, 1.0) * total;
    double _start = .0;

    if (strokeCapRound) {
      _start = asin(stokeWidth/ (size.width - stokeWidth));
    }

    Rect rect = Offset(_offset, _offset) & Size(
        size.width - stokeWidth,
        size.height - stokeWidth
    );

    var paint = Paint()
      ..strokeCap = strokeCapRound ? StrokeCap.round :
StrokeCap.butt
      ..style = PaintingStyle.stroke
      ..isAntiAlias = true
      ..strokeWidth = stokeWidth;

    // 先画背景
    if (backgroundColor != Colors.transparent) {
      paint.color = backgroundColor;
      canvas.drawArc(
          rect,
          _start,
          total,
          false,
          paint
      );
    }
```

```
    // 再画前景，应用渐变
    if (_value > 0) {
      paint.shader = SweepGradient(
        startAngle: 0.0,
        endAngle: _value,
        colors: colors,
        stops: stops,
      ).createShader(rect);

      canvas.drawArc(
          rect,
          _start,
          _value,
          false,
          paint
      );
    }
  }

  @override
  bool shouldRepaint(CustomPainter oldDelegate) => true;

}
```

下面我们来测试一下，为了尽可能多的展示GradientCircularProgressIndicator的不同外观和用途，这个示例代码会比较长，并且添加了动画，建议读者将此示例运行起来观看实际效果，我们先看看其中的一帧动画的截图：

完整的代码：

```
import 'dart:math';
import 'package:flutter/material.dart';
import '../widgets/index.dart';

class GradientCircularProgressRoute extends StatefulWidget {
  @override
  GradientCircularProgressRouteState createState() {
    return new GradientCircularProgressRouteState();
  }
}

class GradientCircularProgressRouteState
```

```
    extends State<GradientCircularProgressRoute> with
TickerProviderStateMixin {
  AnimationController _animationController;

  @override
  void initState() {
    super.initState();
    _animationController =
        new AnimationController(vsync: this, duration:
Duration(seconds: 3));
    bool isForward = true;
    _animationController.addStatusListener((status) {
      if (status == AnimationStatus.forward) {
        isForward = true;
      } else if (status == AnimationStatus.completed ||
          status == AnimationStatus.dismissed) {
        if (isForward) {
          _animationController.reverse();
        } else {
          _animationController.forward();
        }
      } else if (status == AnimationStatus.reverse) {
        isForward = false;
      }
    });
    _animationController.forward();
  }

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

  @override
  Widget build(BuildContext context) {
    return SingleChildScrollView(
      child: Center(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.center,
          children: <Widget>[
            AnimatedBuilder(
              animation: _animationController,
              builder: (BuildContext context, Widget child) {
                return Padding(
                  padding: const EdgeInsets.symmetric(vertical:
16.0),
                  child: Column(
```

```
                    children: <Widget>[
                      Wrap(
                        spacing: 10.0,
                        runSpacing: 16.0,
                        children: <Widget>[
                          GradientCircularProgressIndicator(
                            // No gradient
                            colors: [Colors.blue, Colors.blue],
                            radius: 50.0,
                            stokeWidth: 3.0,
                            value: _animationController.value,
                          ),
                          GradientCircularProgressIndicator(
                            colors: [Colors.red, Colors.orange],
                            radius: 50.0,
                            stokeWidth: 3.0,
                            value: _animationController.value,
                          ),
                          GradientCircularProgressIndicator(
                            colors: [Colors.red, Colors.orange,
Colors.red],
                            radius: 50.0,
                            stokeWidth: 5.0,
                            value: _animationController.value,
                          ),
                          GradientCircularProgressIndicator(
                            colors: [Colors.teal, Colors.cyan],
                            radius: 50.0,
                            stokeWidth: 5.0,
                            strokeCapRound: true,
                            value: CurvedAnimation(
                                    parent: _animationController,
                                    curve: Curves.decelerate)
                                .value,
                          ),
                          TurnBox(
                            turns: 1 / 8,
                            child:
GradientCircularProgressIndicator(
                                colors: [Colors.red, Colors.orange,
Colors.red],
                                radius: 50.0,
                                stokeWidth: 5.0,
                                strokeCapRound: true,
                                backgroundColor: Colors.red[50],
                                totalAngle: 1.5 * pi,
                                value: CurvedAnimation(
                                        parent:
```

```
_animationController,
                                        curve: Curves.ease)
                                    .value),
                            ),
                            RotatedBox(
                              quarterTurns: 1,
                              child:
GradientCircularProgressIndicator(
                                  colors: [Colors.blue[700],
Colors.blue[200]],
                                  radius: 50.0,
                                  stokeWidth: 3.0,
                                  strokeCapRound: true,
                                  backgroundColor:
Colors.transparent,
                                  value: _animationController.value),
                            ),
                            GradientCircularProgressIndicator(
                              colors: [
                                Colors.red,
                                Colors.amber,
                                Colors.cyan,
                                Colors.green[200],
                                Colors.blue,
                                Colors.red
                              ],
                              radius: 50.0,
                              stokeWidth: 5.0,
                              strokeCapRound: true,
                              value: _animationController.value,
                            ),
                          ],
                        ),
                        GradientCircularProgressIndicator(
                          colors: [Colors.blue[700],
Colors.blue[200]],
                          radius: 100.0,
                          stokeWidth: 20.0,
                          value: _animationController.value,
                        ),

                        Padding(
                          padding: const
EdgeInsets.symmetric(vertical: 16.0),
                          child: GradientCircularProgressIndicator(
                            colors: [Colors.blue[700],
Colors.blue[300]],
                            radius: 100.0,
```

```
                            stokeWidth: 20.0,
                            value: _animationController.value,
                            strokeCapRound: true,
                          ),
                        ),
                        //剪裁半圆
                        ClipRect(
                          child: Align(
                            alignment: Alignment.topCenter,
                            heightFactor: .5,
                            child: Padding(
                              padding: const EdgeInsets.only(bottom:
8.0),
                              child: SizedBox(
                                //width: 100.0,
                                child: TurnBox(
                                  turns: .75,
                                  child:
GradientCircularProgressIndicator(
                                    colors: [Colors.teal,
Colors.cyan[500]],
                                    radius: 100.0,
                                    stokeWidth: 8.0,
                                    value:
_animationController.value,
                                    totalAngle: pi,
                                    strokeCapRound: true,
                                  ),
                                ),
                              ),
                            ),
                          ),
                        ),
                        SizedBox(
                          height: 104.0,
                          width: 200.0,
                          child: Stack(
                            alignment: Alignment.center,
                            children: <Widget>[
                              Positioned(
                                height: 200.0,
                                top: .0,
                                child: TurnBox(
                                  turns: .75,
                                  child:
GradientCircularProgressIndicator(
                                    colors: [Colors.teal,
Colors.cyan[500]],
```

```
                                    radius: 100.0,
                                    stokeWidth: 8.0,
                                    value:
_animationController.value,
                                    totalAngle: pi,
                                    strokeCapRound: true,
                                  ),
                                ),
                              ),
                              Padding(
                                padding: const EdgeInsets.only(top:
10.0),
                                child: Text(
                                  "${(_animationController.value *
100).toInt()}%",
                                  style: TextStyle(
                                    fontSize: 25.0,
                                    color: Colors.blueGrey,
                                  ),
                                ),
                              )
                            ],
                          ),
                        ),
                      ),
                    ],
                  ),
                );
              },
            ),
          ],
        ),
      ),
    );
  }
}
```