Flutter 绘制指南 - 妙笔生花

> 这一篇是一个承上启下的部分，主要来正式介绍一下 `CustomPaint` ，这个一直被我们当做 工具人 的存在。 前面 8 篇像是一个纯粹的世界，一张白布，没有干扰。通过这篇，你将知道我们绘制的东西该如何和外界打交道，如何将绘制的组件放置到布局之中。

一、认识 CustomPaint

属性	介绍	类型	默认值
painter	背景画板	CustomPainter?	null
foregroundPainter	前景画板	CustomPainter?	null
size	尺寸	Size	Size.zreo
isComplex	是否非常复杂，来开启缓存	bool	false
willChange	缓存是否应该被告知内容可能在下一帧改变	bool	false
child	子组件	Widget?	null

---

1. 前景画板和背景画板

> 前面我们一直用的是 `painter` 属性，即 背景画板 。前后是相对于 CustomPainter 的子组件而言的。 如果使用 `foregroundPainter` 属性，那么绘制内容将会子子组件的上层。

背景	前景
	

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        title: 'Flutter Demo',
        theme: ThemeData(
          primarySwatch: Colors.blue,
        ),
        home: HomePage());
  }
}

class HomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(),
      body: Container(
        child: CustomPaint(
          // painter: BgPainter(), // 背景
          foregroundPainter: BgPainter(), // 前景
          child: Text(
            "张风捷特烈",
            style: TextStyle(color: Colors.white, fontSize: 20),
          ),
        ),
      ),
    );
  }
}

class BgPainter extends CustomPainter {
  @override
  void paint(Canvas canvas, Size size) {
    canvas.drawPaint(Paint()..color = Colors.red);
  }

  @override
  bool shouldRepaint(covariant CustomPainter oldDelegate) {
    return false;
  }
}
```

---

2. 关于 CustomPainter 中的尺寸

> 当你在一个无约束的布局内，使用 `CustomPaint` 时，你会发现 `CustomPainter#paint` 方法中回调的size 对象为 `size(0,0)` ，比如在 `Scaffold` 中，用画板上绘制回调的尺寸大小的红块，发现什么都没有。

```dart
class HomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(),
      body: _buildChild(),
    );
  }

  Widget _buildChild() {
    final Widget just = CustomPaint(
      painter: BgPainter(), // 背景
    );
    return just;
  }
}

class BgPainter extends CustomPainter {
  @override
  void paint(Canvas canvas, Size size) {
    print(size);
    canvas.drawRect(
        Rect.fromPoints(Offset.zero, Offset(size.width, size.height)),
        Paint()..color = Colors.red);
  }

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

- 解决方案有很多，比如你可以为CustomPaint指定尺寸

```dart
Widget _buildChild() {
  final Widget withSize = CustomPaint(
    size: Size(100,100),
    painter: BgPainter(), // 背景
  );
  return withSize;
}
```

- 为CustomPaint设置一个固定尺寸的组件

```dart
Widget _buildChild() {
  final Widget withChild = CustomPaint(
    painter: BgPainter(),
    child: Container(
      width: 100,
      height: 100,
    ), // 背景
  );
  return withChild;
}
```

- 使用LayoutBuilder获取布局区域

```dart
Widget _buildChild() {

  final Widget withLayoutBuilder = LayoutBuilder(
    builder: _builderByLayout,
  );

  return withLayoutBuilder;
}
Widget _builderByLayout(BuildContext context, BoxConstraints constraints) {
  return CustomPaint(
    size: Size(constraints.maxWidth, constraints.maxHeight),
    painter: BgPainter(), // 背景
  );
}
```

---

3. 关于缓存的两个属性

> 这两个属性没用过，我根据源码文档简单介绍一下它们

- `isComplex`

```dart
合成器包含一个光栅缓存，它保存层的 bitmaps，以避免在每一帧上重复渲染这些层的消耗。
如果没有设置这个标志，那么合成器将应用它自己的触发器来决定这个层是否足够复杂，
是否可以从缓存中获益。

如果 [painter] 和 [foregroundPainter] 都为 null，此标志不能设置为true，
因为在这种情况下该标志将被忽略。
```

- `willChange`

```csharp
栅格缓存是否应该被告知这幅画可能在下一帧中改变。

如果 [painter] 和 [foregroundPainter] 都为 null，此标志不能设置为true，
因为在这种情况下该标志将被忽略。
```

二、认识 CustomPainter

> 这里将会简单看一下 `CustomPaint` 绘制的源码实现，如果看不懂也没有关系，完全不会影响你对绘制的掌握。不过如果可以理解，那当然最好，这样你就更加了解绘制内部是如何运行的。

---

1. CustomPainter#paint 方法回调时机

> 我们知道 `CustomPainter` 是真正进行绘制操作的对象。 `CustomPaint` 作为一个 `SingleChildRenderObjectWidget` 组件，一定有其对应的 `RenderObject` 进行绘制。其实也很容易猜到 `CustomPainter` 对象作为 `CustomPaint` 成员被传入，在对应的 `RenderObject` 中，将由它承担绘制的职能。下面源码说明 `CustomPaint` 是被一个 `RenderCustomPaint` 的 `RenderObject` 绘制的。

```dart
---->[CustomPaint#createRenderObject]----
@override
RenderCustomPaint createRenderObject(BuildContext context) {
  return RenderCustomPaint(
    painter: painter,
    foregroundPainter: foregroundPainter,
    preferredSize: size,
    isComplex: isComplex,
    willChange: willChange,
  );
}
```

> `RenderCustomPaint#paint` 是用于绘制 CustomPaint 组件的。可以看出其中至少调用 `_paintWithPainter` 进行绘制，而 `_paintWithPainter` 方法内调用了传入进去的 `CustomPainter` 对象的 paint 方法，也就是说，前几章实现的 paint 方法，本质上是在这里进行调用的。

```dart
---->[RenderCustomPaint#paint]----
@override
void paint(PaintingContext context, Offset offset) {
  if (_painter != null) {
    _paintWithPainter(context.canvas, offset, _painter!);
    _setRasterCacheHints(context);
  }
  super.paint(context, offset);
  if (_foregroundPainter != null) {
    _paintWithPainter(context.canvas, offset, _foregroundPainter!);
    _setRasterCacheHints(context);
  }
}

void _paintWithPainter(Canvas canvas, Offset offset, CustomPainter painter) {
    late int debugPreviousCanvasSaveCount;
    canvas.save();
     // 略... 断言
    if (offset != Offset.zero)
      canvas.translate(offset.dx, offset.dy);
    painter.paint(canvas, size); // <---- 回调 CustomPainter#paint
        // 略... 断言
    canvas.restore();
  }
```

---

2. CustomPainter#shouldRepaint 方法回调时机

> 在 `RenderCustomPaint` 设置 painter 时，会调用 `_didUpdatePainter` 来判断是否应该更新 可以看到 `newPainter.shouldRepaint(oldPainter)` 返回 true 时，会调用 `markNeedsPaint` 进行重绘。所以一般我们会判断 CustomPainter 中的成员变量是否一致来决定是否重绘，如果都一样，则返回false 不重绘，反之重绘，会再次调用 paint 方法。 这也是为什么建议数据成员都从外界获得原因之一，这样可以很明确知道有哪些成员，容易比较他们是否一致。

```dart
---->[RenderCustomPaint#RenderCustomPaint]----
set painter(CustomPainter? value) {
  if (_painter == value)
    return;
  final CustomPainter? oldPainter = _painter;
  _painter = value;
  _didUpdatePainter(_painter, oldPainter);
}

---->[RenderCustomPaint#_didUpdatePainter]----
void _didUpdatePainter(CustomPainter? newPainter, CustomPainter? oldPainter) {
  // Check if we need to repaint.
  if (newPainter == null) {
    assert(oldPainter != null); // We should be called only for changes.
    markNeedsPaint();
  } else if (oldPainter == null ||
      newPainter.runtimeType != oldPainter.runtimeType ||
      newPainter.shouldRepaint(oldPainter)) {
    markNeedsPaint();
  }
 // 略 ...
}
```

三、CustomPainter 使用注意点

1. 处理 CustomPainter 绘制区域

> 默认 CustomPainter 中的绘制内容 都会被显示在屏幕上 ，比如下面的组件区域是 `100*100` ， 但外面的小圆出界了，仍会被绘制出来，那该如何进行优化呢？ 其实用前面的知识就足以解答： 裁剪矩形域

```dart
@override
void paint(Canvas canvas, Size size) {
  canvas.clipRect(Offset.zero & size); //剪切画布
  canvas.translate(size.width/2, size.height/2);
  _drawArcDetail(canvas);
}
```

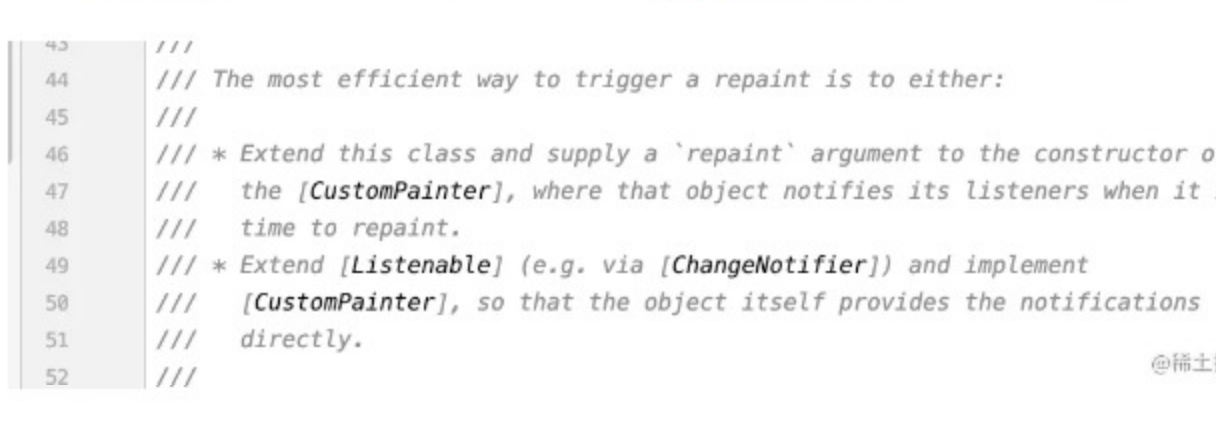
> clipRect 传入一个矩形域，使得之后绘制的内容 只在矩形域中有效 。 如果想显得高大上一点，可以使用 `Offset & Size` 根据偏移和尺寸获取Rect对象 这是由于 `Offset` 重写了 `&` 运算符，源码如下：

```dart
Rect operator &(Size other) => Rect.fromLTWH(dx, dy, other.width, other.height);
```

---

2. CustomPainter 与监听器

这一点 非常重要 ，我是在看 `CupertinoActivityIndicator` 组件源码时发现的这一点。因为 `CustomPainter` 的本身是一个 `Listenable` 子类，可以传入一个 `Listenable对象`，这个对象进行更新时，在会 `shouldRepaint` 允许时触发 `CustomPainter` 的重绘。就 不需要 使用组件状态的 `setState` 来完成画布的刷新。这点在 `CustomPainter` 的源码中也有明确指出：



```
///
/// The most efficient way to trigger a repaint is to either:
///
/// * Extend this class and supply a `repaint` argument to the constructor of
///   the [CustomPainter], where that object notifies its listeners when it is
///   time to repaint.
/// * Extend [Listenable] (e.g. via [ChangeNotifier]) and implement
///   [CustomPainter], so that the object itself provides the notifications
///   directly.
///
```

最高效地触发画板重绘的方式是：

```dart
[1]. 继承自 CustomPainter，在构造函数中对父类 repaint属性 进行赋值，repaint是一个可监听对象，当对象变化
[2]. 继承自 Listenable 实现 CustomPainter，让该类自己执行对自己的更新。
```

> 具体的重绘的使用方法会在下一篇动画中进行演示，这样改交代的就已经交代了，上半场结束，接下来将进入 绘制指南 的下半场。 如果说前面是静态的色彩光怪陆离，那么之后的 动画 、 交互 、 粒子 将是一场 视觉盛宴 。

留言

输入评论（Enter换行，⌘ + Enter发送）

发表评论

全部评论（10）

用户5678780784... JY.3 | 2天前
「将有它承担绘制的职能」

将「由它」承担绘制的职能

这俩词差异还挺大，建议修一下~

点赞 1

> **张风捷特烈**（作者） | 2天前
> 错别字，已更正
> 点赞 回复

用户5678780784... JY.3 | 2天前
第二段 return withSize; 是不是写错了，应该是 withChild ?

点赞 1

> **张风捷特烈**（作者） | 2天前
> 已更正
> 点赞 回复

Yother LV.2 JY.3 | 全干 | 1年前
最后一句话，错了一个字 "具体的重绘的使用方法会在下一篇动画中进行演示，这样改交代的就已经交代了" 中的 改。

点赞 回复

elilai JY.4 | FE | 2年前
第一个例子，前景和背景的图搞反了！！！

1 回复

XiongC JY.3 | 2年前
请问为啥我不用Scaffold，只用SizeBox 或者Padding 包裹就无法裁剪

点赞 1

> **张江** | 2年前
> 没有Scaffold包的时候 paint的绘制区域是全屏，你这设置的300，300 也是不生效的，Scaffold包一下，SizedBox才会生效，指定的区域大小为300*300
> 4 回复

金灿灿 LV.3 JY.4 | 画麦田看日落 | 2年前
万万没想到还有绘制出界这个优化项👍

1 回复

六阿哥来了 JY.3 | iOS @ 无卡数据 | 2年前
大老牛逼

点赞 回复