# 10.1 自定义组件方法简介

当Flutter提供的现有组件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组件，这时我们就需要自定义组件。在Flutter中自定义组件有三种方式：通过组合其它组件、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组件来组合成一个新的组件。例如我们之前介绍的Container就是一个组合组件，它是由DecoratedBox、ConstrainedBox、Transform、Padding、Align等组件组成。

在Flutter中，组合的思想非常重要，Flutter提供了非常多的基础组件，而我们的界面开发其实就是按照需要组合这些组件来实现各种不同的布局而已。

### 自绘

如果遇到无法通过现有的组件来实现需要的UI时，我们可以通过自绘组件的方式来实现，例如我们需要一个颜色渐变的圆形进度条，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组件来绘制出我们期望的外观。我们可以通过Flutter中提供的CustomPaint和Canvas来实现UI自绘。

### 实现RenderObject

Flutter提供的自身具有UI外观的组件，如文本Text、Image都是通过相应的RenderObject（我们将在“Flutter核心原理”一章中详细介绍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代表组件的绘制上下文，通过PaintingContext.canvas可以获得Canvas，而绘制逻辑主要是通过Canvas API来实现。子类需要重写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 API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，在任何需要自定义组件的场景下，我们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，优点是强大灵活，理论上可以实现任何外观的UI，而缺点是必须了解Canvas API细节，并且得自己去实现绘制逻辑。

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，并且Flutter中很多基础组件都是通过RenderObject的形式来实现的，所以后续我们只介绍CustomPaint和Canvas的方式，读者如果对自定义RenderObject的方法好奇，可以查看Flutter中相关基础组件对应的RenderObject的实现源码，如RenderParagraph或RenderImage。