1

在 Flutter 中，Widget 是整个视图描述的基础，在 Flutter 的世界里，包括应用、视图、视图控制器、布局等在内的概念，都建立在 Widget 之上，**Flutter 的核心设计思想便是一切皆 Widget**。

2

Widget 是组件视觉效果的封装，是 UI 界面的载体，因此我们还需要为它提供一个方法，来告诉 Flutter 框架如何构建 UI 界面，这个方法就是 build。

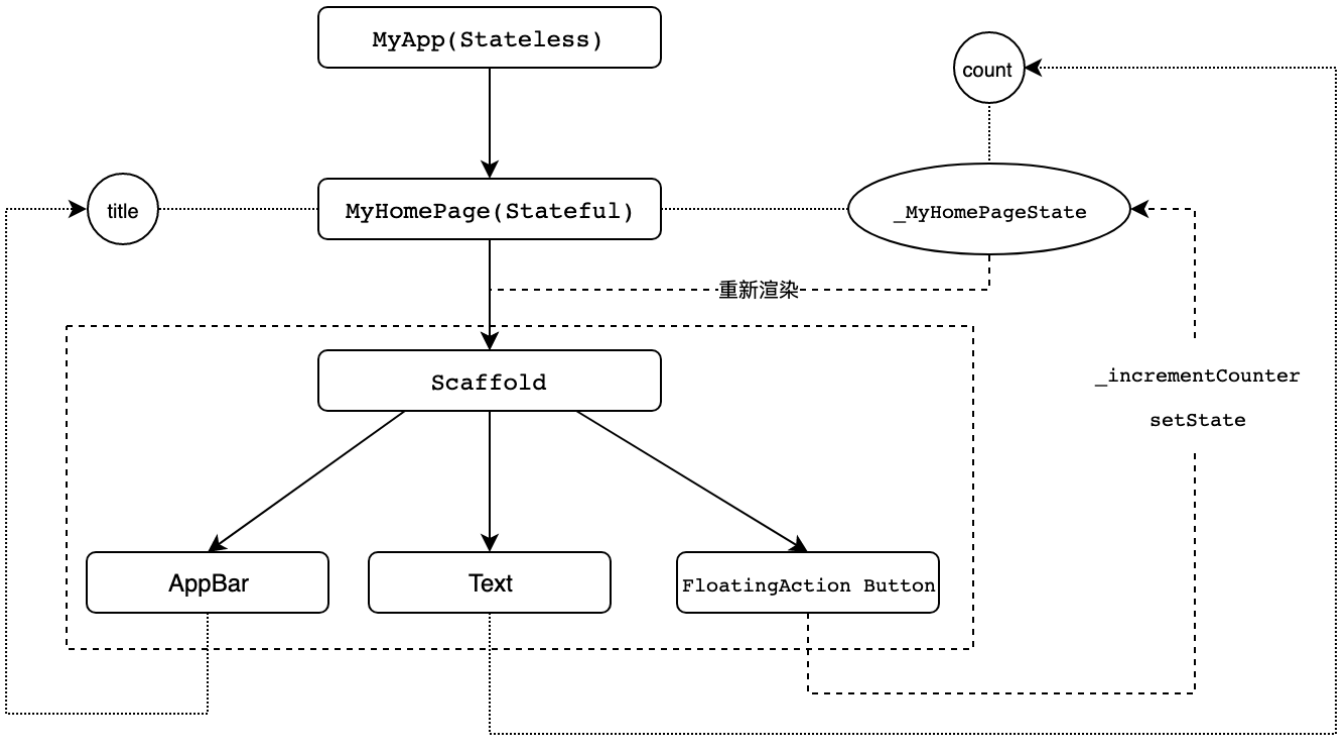
3

StatefulWidget 与 StatelessWidget 的接口设计，为什么会有这样的区别呢?

这是因为 Widget 需要依据数据才能完成构建，而对于 StatefulWidget 来说，其依赖的数据在 Widget 生命周期中可能会频繁地发生变化。由 State 创建 Widget，以数据驱动视图更新，而不是直接操作 UI 更新视觉属性，代码表达可以更精炼，逻辑也可以更清晰。

4

setState 方法是 Flutter 以数据驱动视图更新的关键函数，它会通知 Flutter 框架：我这儿有状态发生了改变，赶紧给我刷新界面吧。而 Flutter 框架收到通知后，会执行 Widget 的 build 方法，根据新的状态重新构建界面。



5

State 通过调用 build 方法，以相应的数据配置完成了包括导航栏、文本及按钮的页面视图的创建。

这里需要注意的是：状态的更改一定要配合使用 setState。通过这个方法的调用，Flutter 会在底层标记 Widget 的状态，随后触发重建。

6

Widget 只是视图的“配置信息”，是数据的映射，是“只读”的，并不负责渲染。对于 StatefulWidget 而言，当数据改变的时候，我们需要重新创建 Widget 去更新界面，这也就意味着 Widget 的创建销毁会非常频繁。

为此，Flutter 对这个机制做了优化，其框架内部会通过一个中间层去收敛上层 UI 配置对底层真实渲染的改动，从而最大程度降低对真实渲染视图的修改，提高渲染效率，而不是上层 UI 配置变了就需要销毁整个渲染视图树重建。这样一来，Widget 仅是一个轻量级的数据配置存储结构，它的重新创建速度非常快，所以我们可以放心地重新构建任何需要更新的视图，而无需分别修改各个子 Widget 的特定样式。