

Flutter 底层进阶篇

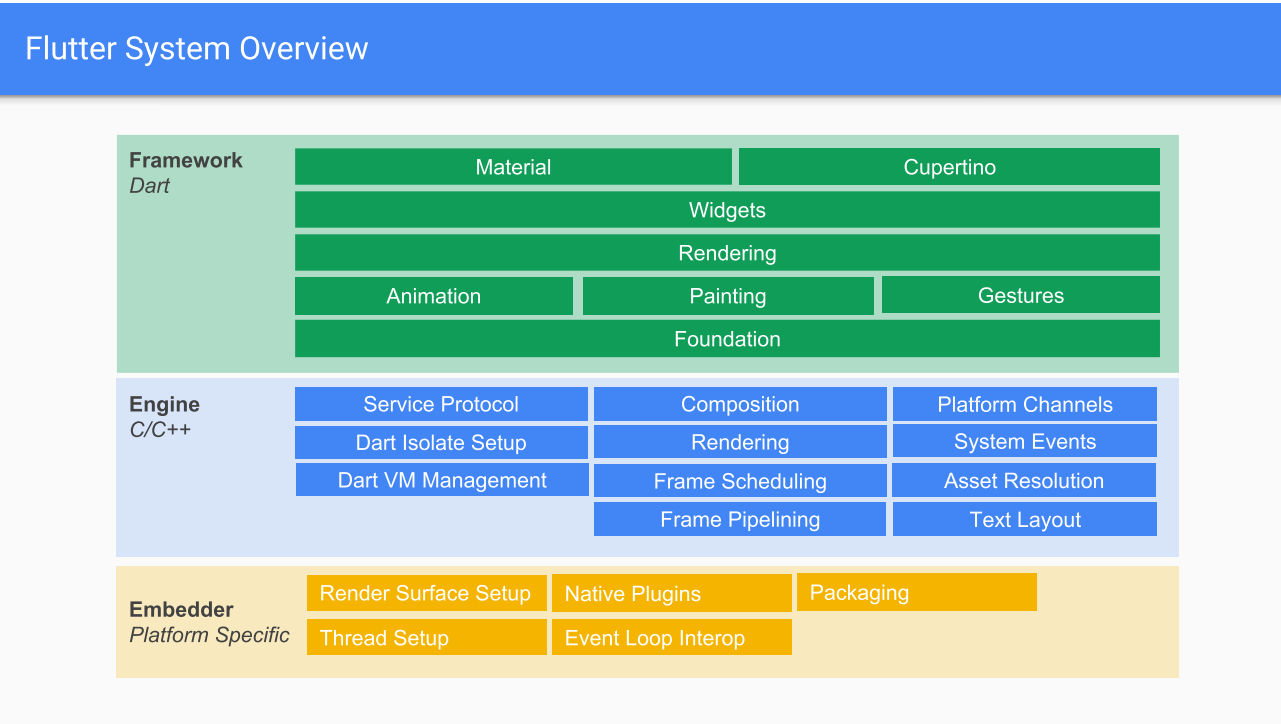
经过前面的一系列学习，应该在广度上对 Flutter 有了比较全面的认识，但我们也需要在深度上了解一下 Flutter。

所以接下来主要讲 Flutter 底层的实现，例如架构等，虽然我们更关心如何用 Flutter 写 APP，但是理解 Flutter 的架构，可以让我们在较高层面理解 Flutter 的工作方式和原理，从而知道 Flutter 可以实现哪些功能及无法实现哪些功能。

有关 Flutter 架构的内容，建议可以结合前面的章节一起来看，会有更深的理解。

Flutter 架构深度解析

Flutter 框架是一个多层架构，每层构建在前一层之上，下图展示了 Flutter 架构的各个部分：



总共有三层，从下到上依次为：

1. Embedder

Embedder 是平台指定的语言实现，主要处理平台相关的操作，是为了 Flutter 能适配各种平台的嵌入层。Embedder 有 Android 的实现，也有 iOS 的实现，也有 Linux、MacOS、Windows 的实现。

2. Engine

Engine 层由 C/C++ 实现，Flutter Engine 为 Flutter 应用提供了运行环境，是 Flutter 的核心。

3. Framework

Framework 层由 Dart 实现，是 Flutter 开发的框架，开发 Flutter 的 APP，大部分时间都是和这一层打交道。

Flutter 架构 -- Engine

Engine 层由 C/C++ 实现，Flutter Engine 为 Flutter 应用提供了运行环境，是 Flutter 的核心。

上图是 Engine 层的截图，Engine 下面标了 C/C++，表示了 Engine 层由 C/C++ 实现，而且可以看到 Engine 层包含了很多的功能，我把它归纳如下：

- 渲染相关：Composition 、 Rendering 、 Frame Scheduling 、 Frame Pipelining
- Dart 相关：Service Protocol 、 Dart Isolate Setup 、 Dart VM Managemnt
- 平台通道：Platform Channels
- 系统事件：System Events

- 资源解析：Asset Resolution
- 文字渲染：Text Layout

其中的核心功能，包括动画和图形渲染，文件和网络 I/O，Platform Channels，插件体系结构，以及 Dart 的运行时环境和编译工具链等。想要深入了解 Flutter Engine 的话，可以在 [Github \(https://github.com/flutter/engine\)](https://github.com/flutter/engine) 上查看源码。

接下来介绍一些核心的东西。

渲染引擎：Skia

Skia 是 Flutter 的图形引擎，是 Flutter 渲染过程中的重要一环。

Skia 是 Google 的跨平台 2D 向量图形库，而且已经发展的很成熟：在 2005 年被 Google 收购后，已经成为 Google Chrome，Chrome OS，Android，Mozilla Firefox，Firefox OS 等众多产品的图形引擎，支持的平台包括 Windows7+，macOS 10.5+，iOS8+，Android4.1+，Ubuntu14.04+ 等，并且 Skia 发展的已经很稳定了。

Android 是自带 Skia，所以 Android 端 Flutter 不需要打包 Skia，所以在 Android 端，Flutter 的包大小会小一些；而 iOS 不带 Skia，所以 iOS 的 Flutter 包会大一些。

Dart 运行时环境

Dart 运行时环境包括 Dart VM 及其他 Dart 运行所需要的库。

- Dart VM

Dart VM 除了实现普通 Dart 的核心库之外，还增加了一个 `dart:ui` 库，这个库是专门为 Flutter 定制的，提供 Skia 和 Shell 功能的低级 API。

而且在 Flutter Debug 模式中，Flutter Engine 使用的是 Dart VM，而 Dart VM 支持 JIT（即时编译），从而使 Flutter 在 Debug 阶段有 Hot Reload 的功能。

- Dart 相关库

在 Release 模式下，Flutter 不会带 Dart VM，因为 Flutter 使用的是 AOT（静态编译），会编译成 Native Arm Code，所以不再需要 Dart VM，但是运行 Native Arm Code 也需要其他 Dart 相关库，例如 Garbage Collection (GC，垃圾回收) 等，这些库不包含在 Dart VM 里。

讲到这里，我有一个问题：为什么 Dart 可以运行在不同的平台上？

是因为 Dart 并不是直接运行在平台上，而是运行在 Flutter Engine 上，Flutter Engine 为 Dart 提供了运行环境。

Platform Channel

Platform Channel 是平台通道，用于 Flutter 与 Native 通信：

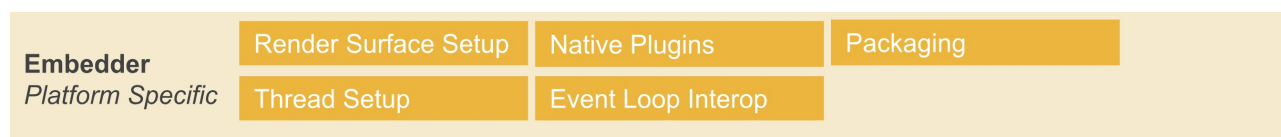
- 应用的 Flutter 部分通过平台通道（platform channel）将消息发送到 Native（iOS 或 Android）。
- Native 接收消息，然后调用 Native 的代码处理，然后将响应发送回 Flutter。

Platform Channel 是一个很重要的功能，尤其是 Flutter 与 Native 混合开发的时候，后面会具体讲到怎么使用。

Flutter 架构 -- Embedder

Embedder 是平台指定的语言实现，主要处理平台相关的操作，是为了 Flutter 能适配各种平台的嵌入层。Embedder 有 Android 的实现，也有 iOS 的实现，也有 Linux、MacOS、Windows 的实

现。



上图是 Embedder 层的截图，Embedder 下面标了 Platform Specific，表示 Embedder 是基于平台实现的，包含的功能有：

- Render Surface Setup：渲染设置
- Native Plugins：平台的插件
- Packaging：包装 Flutter AOT 的产物：Native Arm Code，使 Native Arm Code 在平台上运行。
- Thread Setup：Flutter 运行线程设置
- Event Loop Interop：Flutter 事件循环

可以看到主要都是底层的机制，而且都是和平台相关的实现，除此之外 Embedder 还有另一个很重要的功能，就是 Embedder 是 Flutter Engine 和 Platform（平台）之间交互的桥梁。

Embedder API：交互的桥梁

Flutter Engine 和 Platform（平台）靠什么交互呢？

就是 **Embedder API**。

在 Flutter Engine 和 Platform 中间有一层 API，就是 [Embedder API](https://github.com/flutter/engine/tree/master/shell/platform)
(<https://github.com/flutter/engine/tree/master/shell/platform>)

Flutter Engine 通过这些 Embedder API 调用平台的能力，而平台通过实现这些 Embedder API，就可以在不同的平台上运行 Flutter Engine。

Shell：Embedder API的实现

实现 Embedder API 的叫做 **Shell**，Google 为 Flutter 实现了不同平台的 Shell，比如就有 Android Shell 和 iOS Shell，也有 Linux Shell、MacOS Shell、Windows Shell，这些 Shell 在不同平台上，用平台指定的语言实现，并提供相关 IME（例如：屏幕）和系统应用程序声明周期事件的通信、渲染、插件、线程创建和管理、事件循环等。

Flutter 架构 -- Framework

Framework 层由 Dart 实现，是 Flutter 开发的框架，开发 Flutter 的 APP，大部分时间都是和这一层打交道。所以之后的章节大部分讲的都是 Framework 层的内容。

上图是 Framework 层的截图，Framework 下面标了 Dart，表示了 Framework 层由 Dart 语言实现，Framework 层也是一个多层架构，从上到下分别是：

1. Foundation（基础库层）

Foundation 库即是 `dart:ui` 库，为 Flutter 提供了基本的类和函数，包括处理与 Flutter Engine 层的通信，以及用于 Flutter 框架的最低级别服务，例如驱动输入、图形文本、布局和渲染等。

使用这一层的功能也能构建 Flutter APP，但是因为这一层没有封装，你只能手动计算布局坐标，手动捕捉用户输入和混合动画，用 Foundation 去写 APP 将非常复杂，成本也很大。所以需要 Foundation 层进行封装。

如果想要知道 Foundation 层提供的类和函数的可以查看 [这里]

(<https://docs.flutter.io/flutter/foundation/foundation-library.html>)。

2. Rendering（渲染层）

渲染层包括 Rendering、Animation、Painting、Gestures，是 Foundation 之上的第一个抽象层，是对 Foundation 层的封装。

这一层主要是完成 UI 的布局和绘制，为了优化这一复杂的过程，采用了智能的算法去缓存那些昂贵的计算，使得每次的迭代量最小，可以提高渲染和绘制的性能。

3. Widget（组件层）

Widget 就是 Flutter UI 的基本元素，提供了可以在 Flutter 中使用的 UI 组件，是我们实际开发中最常用的元素，所有的 Widget 可以分成以下四类：

1. Structural Widget（布局组件），例如 Column 和 Row 等，控制布局的。
2. Visual Widget（绘制组件），例如 Text 和 Image 等，在屏幕上显示内容的。
3. Interaction Widget（交互组件），例如 GestureDetector 等，处理用户手势的。
4. Platform Widget（平台组件），例如 AndroidView 等，将平台的 View 嵌入到 Flutter 中使用的

我们也可以用这些组件实现自己的组件，但是是通过组合的方式，因为在 Flutter 中，组合大于继承。

4. Material 和 Cupertino 风格的组件

Flutter 为了减轻开发人员的工作量，实现了两种不同风格的组件：Material 和 Cupertino。Material 用于 Android，Cupertino 用于 iOS。有了这些组件，开发人员不需要再做额外的工作，就可以让 Flutter 的 UI 风格适应不同的平台，让 Flutter UI 获得和 Native UI 一样的使用体验。

总之，层级越高，更易于处理，但低层级的可以提供更复杂的细粒度的控制。

想阅读 Framework 的源码，可以阅读 [Github](https://github.com/flutter/flutter/tree/master/packages/flutter)
(<https://github.com/flutter/flutter/tree/master/packages/flutter>) 上的源码。