

了解 Flutter 的 Thread Model（线程模型）

Flutter Architecture overview

从图中可以看出 Flutter 有3部分组成：

1. Framework(Dart)：我们在写 Flutter 应用中，直接接触的
2. Engine(C/C++)：引擎层，提供核心技术，例如
 - Skia：2D图形渲染库
 - Dart：一个用于垃圾收集的面向对象语言的 VM,DartVM 有自己的线程池
 - shell：不同的平台有不同的 shell，例如有 Android和 iOS 的 shell
3. Embedder(Platform Specific)：嵌入层，为 Engine 创建和管理线程，作用是把 Engine 的 task runners（任务运行器）运行在嵌入层管理的线程上。

实现 Isolate 的线程由 Flutter 创建和管理，除了实现 Isolate 的线程，Flutter 还有其余的线程，本篇文章探讨 Flutter 的 threading model（线程模型）。

创建和管理线程

在 Flutter Engine 中定义了四种 Task Runners（任务运行器），Task Runners 是需要运行在平台提供的线程上的，所以 Flutter Engine 是不创建和管理线程的，是由 Embedder 为 Flutter Engine 创建和管理线程（包括线程里的消息循环），这四种 Task Runners 分别是：

- Platform Task Runner
- UI Task Runner
- GPU Task Runner
- IO Task Runner

理解这四种 Task Runner，可以让我们知道我们写的代码是运行在哪个线程上，从而知道可以做什么，不可以做什么。

Platform Task Runner

所在的线程

Platform Task Runner 运行所在的线程叫 Platform Thread，Platform Thread 必须要运行在平台的主线程上。

Android iOS

主线程 主线程

功能

处理平台（Android/iOS）的消息

Q&A

- 为什么一定要是平台的主线程？

因为 Platform Task Runner 的功能是要处理平台的消息，但是平台的 API 都是只能在主线程调用，所以 Platform Task Runner 运行所在的平台的线程必须是主线程

- 一个 Flutter Engine 有一个还是多个 Platform Thread?

一个 Flutter Engine 对应一个 Platform Thread (一个 Flutter 应用启动的时候会创建一个 Engine 实例，Engine

创建的时候会创建一个 Platform Thread 供 Platform Task Runner 使用)

- 如果阻塞 Platform Thread 会有什么后果?

阻塞 Platform Thread 不会直接导致 Flutter 应用的卡顿 (跟 iOS、Android 主线程不同)。尽管如此，也不建议在这个 Runner 执行繁重的操作，长时间卡住 Platform Thread 应用有可能会被系统 Watchdog 强杀。

UI Task Runner

所在的线程

UI Task Runner 运行所在的线程叫 UI Thread，UI Thread 对必须运行在平台的哪个线程没有特别要求，但在 Android 和 iOS 上都是子线程。

Android iOS

子线程 子线程

这里很容易让大家误会，因为一看名字，UI Thread，第一反应，UI ? 那肯定是主线程，其实并不是，因为这里的 UI 指的是 Flutter 的 UI，Flutter 的 UI 不需要在平台的主线程里渲染，相反 Flutter UI Thread 为了不阻塞平台的主线程，提高 Flutter UI 渲染的性能，反

而创建了新的线程，所以在 Android 和 iOS 上，UI Task Runner 都是运行在子线程的，所以 UI Task Runner 可以理解成是 Flutter 的主线程，但却运行在平台的子线程上。

功能

- 1) 用于执行 Dart Root Isolate 代码
- 2) 渲染逻辑，告诉 Engine 最终的渲染
- 3) 处理来自 Native Plugins 的消息
- 4) timers
- 5) microtasks
- 6) 异步 I/O 操作(sockets、file handles 等)

Q&A

- UI Task Runner 和 Root Isolate 的关系？

Isolate 是 Dart 里的概念，它是类似于线程(thread)但不共享内存的独立运行的 worker，是一个独立的 Dart 程序执行环境。在 Flutter 中默认执行 Dart 代码的就是 Root Isolate。

Root Isolate 是运行在 UI Thread 上的，所以这就是为什么 isolate 就可以理解为单线程，而且 UI Thread 上有和平台主线程一样的 event loop 架构，是为了处理 Root Isolate 里的消息。

- 如果阻塞 UI Thread 会有什么后果？

阻塞 UI Thread 会直接导致 Flutter 应用卡顿掉帧。

GPU Task Runner

所在的线程

GPU Task Runner 运行所在的线程叫 GPU Thread，GPU Thread 对必须运行在平台的哪个线程没有特别要求，但在 Android 和 iOS 上是子线程。

Android iOS

子线程 子线程

功能

GPU Task Runner 主要用于执行设备 GPU 的指令。在 UI Task Runner 创建 Layer Tree，在 GPU Task Runner 将 Layer Tree 提供的信息转化为平台可执行的 GPU 指令。

Q&A

- UI Task Runner 和 GPU Task Runner 都有负责渲染，他们是跑在同一个子线程上吗？

UI Task Runner 和 GPU Task Runner 跑在不同的线程。GPU Runner 会根据目前帧执行的进度去向 UI Task Runner 要求下一帧的数据，在任务繁重的时候可能会告诉 UI Task Runner 延迟任务。这种调度机制确保 GPU Task Runner 不至于过载，同时也避免了 UI Task Runner 不必要的消耗。

- 如果阻塞 GPU Thread 会有什么后果？

在此线程耗时太久的话，会造成 Flutter 应用卡顿，所以在 GPU Task Runner 尽量不要做耗时的任务，例如加载图片的时候，去读取图片数据，就不应该放在 GPU Task Runner，

而是放在接下来要讲的 IO Task Runner。从性能考虑，建议为 GPU Task Runner 单独开一个线程。

IO Task Runner

所在的线程

IO Task Runner 运行所在的线程叫 IO Thread，IO Thread 对必须运行在平台的哪个线程没有特别要求，但在 Android 和 iOS 上是子线程。

Android iOS

子线程 子线程

功能

1) 主要功能是从图片存储（比如磁盘）中读取压缩的图片格式，将图片数据进行处理为 GPU Runner 的渲染做好准备。IO Runner 首先要读取压缩的图片二进制数据（比如 PNG，JPEG），将其解压转换成 GPU 能够处理的格式然后将数据上传到 GPU。

2) 加载其他资源文件

Q&A

- 如果阻塞 IO Thread 会有什么后果？

阻塞 IO Thread，不会影响 Flutter UI 的渲染，所以阻塞 IO Thread 不会影响 Flutter，但加载图片和资源的时候可能会延迟，所以从性能角度考虑，还是建议为 IO Task Runner 单独开一个线程。

各个平台的线程配置

iOS

为每个引擎实例的 UI，GPU 和 IO 任务运行程序创建专用线程。所有引擎实例共享相同的 Platform Thread 和 Platform Task Runner。

Android

为每个引擎实例的 UI，GPU 和 IO 任务运行程序创建专用线程。所有引擎实例共享相同的 Platform Thread 和 Platform Task Runner。

Fuchsia

每一个 Engine 实例都为 UI，GPU，IO，Platform Runner 创建各自新的线程。

Flutter Tester (used by flutter test)

UI，GPU，IO 和 Platform 任务运行器使用相同的主线程。

实现Embedder API 的平台

Android/iOS

Flutter 默认实现的是 Android 和 iOS 平台的 Embedder API。

PC

在 PC 平台上有两种方式实现了 Flutter Embedder API：

1. 用 C++ 实现的 Flutter Embedder API

GitHub 上的[flutter-desktop-embedding](https://github.com/google/flutter-desktop-embedding)
(<https://github.com/google/flutter-desktop-embedding>)

2. 用 Go 实现的 Flutter Embedder API

GitHub 上的[go-flutter-desktop-embedder](https://github.com/Drakirus/go-flutter-desktop-embedder)
(<https://github.com/Drakirus/go-flutter-desktop-embedder>)