

移动端跨平台开发方案的演进

我们先来了解一下移动端跨平台开发方案的演进历史。

从 Android、iOS 推出至今，已有十几年，移动端的开发技术也在不断发展，最开始的时候，都是使用原生开发，但却有一个明显的痛点，就是相同的功能需要在不同的平台上都实现一遍，所以就有了一个很迫切的需求，能否只需要写一次代码，就可以在各个端都运行？

本节就回顾一下跨平台开发方案的演进历史，我们根据跨平台方案所用的技术分为以下几种：

WebView 流

第一种方案，我们称之为 WebView 流。因为 WebView 作为一个显示 web 页面的容器，它本身是跨平台的，所以利用 WebView 跨平台属性的技术方案就称之为 WebView 流。WebView 流的发展也经历了如下的过程：

1. 纯 H5

最开始的时候，是在 APP 里嵌入一个 WebView，负责显示 H5，可能是独立的一个页面，也有可能是 Native 页面的一部分。

但在这个阶段，一方面，WebView 的性能没有 Native 的好，另一方面，H5 页面和 Native 的页面几乎是没有交互的，使得 Native 的开发和 H5 的开发是彼此独立的，Native 无法使用 H5 的特性，H5 也无法使用 Native 的特性，这些都限制了 WebView 的使用场景。

2. Hybrid

为了打破 Native 和 H5 割裂的情况，就出现了 JSBridge，JSBridge 是 Native 代码与 JavaScript 代码的通信桥梁。

JSBridge 的出现，使得 H5 可以使用 Native 的能力，Native 也能使用 H5 的能力，使 Native 和 H5 能完美融合在一起，出现了一系列的 Hybrid 开发框架，比如 PhoneGap 等，这些架构具有开发成本低、简单、跨平台的优点。但是这些 Hybrid 框架都是基于 WebView 实现的，所以无法避免 WebView 本身的劣势：内存占用多、网页加载速度慢、渲染慢、JavaScript 执行慢等，因为这些性能问题，所以大多数人在开发过程中，主要页面都是使用 Native 开发，只有少数页面才采用 Hybrid 的框架。

3. 基于 Hybrid 的优化改进方案

针对具体的性能问题，开始有了不同的优化方案，比如：网页加载速度慢，则采用离线包的方案；网页渲染效率低，则优化 dom 树等。虽然性能上有很大提升，但是始终绕不过两大问题，第一个是，WebView 的渲染性能比 Native 差，第二个就是，JavaScript 是解释执行语言，运行效率也比 Native 差。所以无论怎么优化，使用 Hybrid 开发的页面的性能肯定不会比 Native 的好。所以，如果是你，你会在自己的 APP 中全部采用 Hybrid 吗？答案是肯定不会。

ReactNative 流

第二种是 ReactNative 流，ReactNative 流是从 WebView 流发展而来，典型的方案是 ReactNative 和 Weex，这两种方案，都抛弃了 WebView 这个累赘，但仍然使用 H5 的技术栈开发，使用 JavaScript 开发，那是如何做到跨平台的呢：

跨平台的布局引擎

ReactNative 内置了跨平台的布局引擎，可以将 H5 的布局转化为 Native 的布局。

使用 Native 原生组件渲染

将 ReactView 组件使用 Native 原生组件渲染。

JavaScript 引擎

ReactNative 内置 JavaScript 的引擎，从而可以在不同平台上运行 JavaScript 代码。

ReactNative 流采用 JavaScript + JavaScript 引擎+ Native 的技术方案，利用了 JavaScript 的跨平台特性实现了移动端的跨平台方案。

编译流/虚拟机流

第三种是编译流/虚拟机流，这种的方案是 [Xamarin](https://baike.baidu.com/item/Xamarin) (<https://baike.baidu.com/item/Xamarin>)。这种方案大家可能不熟悉，因为 Xamarin 在国内用的很少，但在国外用的比较多。

Xamarin 使用 C# 开发，在 iOS 平台，会将 C# 编译成可以直接在 iOS 上运行的 Native ARM Code，在 Android 平台上，C# 代码运行在 Mono 虚拟机上，而 Mono 虚拟机运行在 Android 上。把这种通过第三种语言开发，编译到可以直接在目的平台上运行，或者运行在虚拟机上的方式就叫编译流/虚拟机流。

可能有的人会有疑问，既然可以运行在虚拟机上，那么 iOS 上也运行虚拟机不就可以了，为何采用的是编译？我猜测，如果在 iOS 上运行虚拟机，那么就意味着 iOS 上可以热更新代码，这个是不被苹果应用商店所允许的，所以在 iOS 平台上采用的是编译流。

Xamarin 的 UI 渲染也是映射成 Native 原生组件来渲染的。

游戏引擎流

第四种是游戏引擎流，例如 Unity 等。如果只看游戏引擎的技术，用做跨平台开发在适合不过了，因为逻辑和渲染都是游戏引擎自己负责的，但是用游戏引擎开发普通 APP，就是杀鸡用牛刀。

游戏引擎也不适合用来开发普通 APP，因为：

1. 游戏引擎的安装包体积都很大。
2. 游戏引擎和普通 APP 的刷新机制不同，游戏引擎是实时刷新，普通 APP 是按需刷新，而且实时刷新也造成耗电量较高。
3. 游戏引擎虽然能做出复杂的游戏界面，但是实现 Android 或 iOS 的原生界面，却需要自己在重新开发。
4. 而且游戏引擎适合开发一个新的 APP，不适合和原生 APP 做混合开发。

理想的跨平台开发方案

介绍了这么多种跨平台的方案，不难看出，实现跨平台方案需要解决的技术难题：

- 一个是逻辑代码：逻辑代码需要用一种语言来实现。
- 一个是 UI：要么自己渲染，要么使用原生的渲染。

Webview 流使用 JS 开发，JS 逻辑代码运行在 Webview 容器里，UI 也是使用的 Webview 容器自己的渲染；ReactNative 流使用 JS 开发，JS 逻辑代码运行在 JS 引擎上，使用原生的渲染；Xamarin 使用 C# 开发，将 C# 编译后在 iOS/Android 上运行，UI 使用的原生的渲染；游戏引擎，逻辑代码和 UI 都是运行在游戏引擎的容器里。

假设让你设计一个新的跨平台解决方案，先不考虑技术选型和实现，你心中的跨平台技术方案应该有哪些特点？

如果是我的话，我心中的跨平台技术方案是这样的：

1. 只使用一种语言开发。
2. UI 渲染最好是自己来渲染，有自己的一套渲染引擎，因为这样最省心，而且渲染性能要和 Native 一样。
3. 能和平台很方便的进行交互。

可见如何将逻辑代码和UI跨平台的实现，如何和平台交互，是实现一个跨平台方案的关键。

跨平台方案的新发展方向 —— Flutter

现在跨平台方案有了新的发展方向，就是 Flutter。

Flutter 开辟了一种全新的思路，利用 Dart 语言，同时支持 JIT 和 AOT 两种编译方式的特性，在不同场景下使用不同的编译方式，达到最高效的开发和运行体验。

在 Debug 模式下，为了保证开发体验，采用 JIT 这种动态编译的方式，将代码运行在 Dart 虚拟机上，使得我们编写的代码可以实时更新，实现 HotReload 的特性，提升开发体验。

而在 Release 模式下，又需要保证运行速度和渲染流程度，则会采用 AOT 的编译方式，将代码直接编译成各自平台的 Native 代码，以此提高使用体验。

在 UI 渲染方面，Flutter 的渲染不依赖于平台，基于自带的 Skia 渲染引擎，构建了一套完整的跨平台 UI 渲染框架；在和平台交互方面，Flutter 提供了 Platform Channel 的通道，可以方便的和 Native 交互。可以说是 Google 融合多种技术才有的产物，也是跨平台方案发展的必然产物。

除此之外，一个好的技术，还需要有完善的文档去让人学习，有专业的人去维护，让技术不断发展，而在这方面，Flutter 即有完善的文档，也有 Google 团队的维护。因此 Flutter 肯定会更好的发展。

Flutter 与其他跨平台方案的对比

经过上面的介绍，以跨平台作为 X 轴，以性能作为 Y 轴，就可以得出如下的图：

Flutter 在跨平台和性能上都是比其他方案要好。

Flutter：为未来而生

Flutter 不仅是一个跨平台的 UI 框架，也是 Google 的新一代操作系统 Fuchsia 的 UI 框架。

Fuchsia 是 Google 开发的下一代操作系统。和以前 Google 开发的操作系统，如基于 Linux 内核的 Chrome OS 和 Android 等不同，Fuchsia 基于新的名为 Zircon 的微内核，Fuchsia 的设计目标之一是可运行在众多的设备上，包括汽车的娱乐媒体系统和嵌入式设备，如红绿灯、数字手表、智能手机、平板电脑与个人计算机。

根据 [Bloomberg](https://www.bloomberg.com/news/articles/2018-07-19/google-team-is-said-to-plot-android-successor-draw-skepticism)

(<https://www.bloomberg.com/news/articles/2018-07-19/google-team-is-said-to-plot-android-successor-draw-skepticism>) 2018 年 7 月 19 号的一篇文章，Fuchsia 团队计划在 3 年内首先将 Fuchsia OS 用于智能设备，5 年内将 Fuchsia OS 搭载在智能手机和个人计算机上。在 2018 年 10 月，有报道称 Google Home Hub 将搭载 Fuchsia OS，代号为 Astro，可见 Fuchsia OS 离我们应该不远了。

所以使用 Flutter 开发，不仅可以运行在 Android 和 iOS 上，也可以运行在 Fuchsia 上，不仅可以用于现在，也可以用于未来。还有什么理由不来学习呢？