设计研究与应用

一种基于 ReactNative 的代码压缩与动态加载的实现方法

金昱东

（北京邮电大学 网络技术研究院 北京 100876）

摘 要：随着移动互联网的飞速发展，移动设备与平台也在不断更新迭代，多种平台与操作系统共存。对用户而言，多了一种选择，而对于开发者而言，则大大增加了开发成本。为了实现相同的功能，为每个平台单独开发应用无疑是一种非常耗费人力资源的解决方案；而基于 Web 的跨平台开发方案并不能保证在不同平台提供统一流畅的用户体验。因此降低开发成本、保证应用体验是移动平台开发一直追求和关注的两个核心点。本文论述了 React Native 相比其他跨平台解决方案的优势与不同，并在 React Native 基础上提供了一种代码压缩与动态加载的解决方案，大幅减少动态加载业务代码的大小与传输时间，将其在实际开发中的实用性进一步提高。

关键词：软件工程；移动互联网；跨平台；ReactNative；动态加载

中图分类号：TP311 **文献标识码**：A **DOI**：10.3969/j.issn.1003-6970.2016.02.020

本文著录格式：金昱东. 一种基于 ReactNative 的代码压缩与动态加载的实现方法[J].软件，2016，37（02）：81-84

Code Compressing and Dynamic Loading Implementation Based on ReactNative

JIN Yu-dong

（*Institute of Network，Beijing University of Posts and Telecommunications，Beijing* 100876，*China*）

【Abstract】: With the rapid evolution of internet，mobile devices and platforms are updating and iterating more frequently than ever，multiple platforms coexist.It benefits the common users but not the developers，because multiple platforms also mean even more resources to invest in development.In the meantime，web based development framework cannot assure a unified experience in different platforms.Reduce the cost of development and the assurance of user experience is the two key points that developers are pursuing.This paper introduce a new multiple platform development framework called React Native which is able to assure both of the two key points，further this paper comes up with a optimization that can vastly reduce the transmission and loading time of the execution of React Native.The optimization increase user experience and make the framework more practicability in real development.

【Key words】: Software engineering；Mobile internet；Multi-platform；React native；Dynamic loading

0 引言

移动端的操作系统目前呈现两极分化，Android 与 iOS 是目前最普及的两个移动操作系统。然而在为它们开发同一个应用程序时，通常都需要两套开发人员来维护两个不同的代码仓库。除了 React Native 外比较流行的解决办法是使用 Html5[1]等 Web 技术[2]来实现“一套代码，多处运行”。然而由于设备与操作系统的差异性，基于 Web 的界面与交互在两个平台上的体验并不一致且相差甚远[3]。因此它只解决了开发成本的问题，并没有解决保证应用体验的问题。然而 React Native 则很好的同时解决了这两个问题。

另外值得一提的是在 Android 与 iOS 平台都可以实现代码的动态加载，与 Web 同样灵活。即在程序运行的过程中更新业务代码，可以用来实现 bug 修复或应用升级等功能。

Android 依托于 Java 的 Classloader，而 iOS 依托于 Objective-的动态性。React Native 主要使用 JS 作为开发语言，也非常好的支持了动态加载机制，但缺点

作者简介：金昱东（1991-），男，研究生，网络测量、移动互联网

是打包后的 JS 代码过多，不利于在质量较低的移动网络下进行传输。因此本论文提出了一种基于 React Native 的代码压缩方法，可以大幅精简 JS 代码量，使其在移动网络下的稳健性更高、实用性更强。

1 React Native 的概念

React Native[4]是 Facebook 在 2015 年发布的一套基于 JavaScript 的开源框架。React Native 结合了 Web 应用和 Native 应用的优势，可以使用 JavaScript 来开发 iOS 和 Android 原生应用。在 JavaScript 中用 React[5]抽象操作系统原生的 UI 组件，代替 DOM 元素来渲染界面。因此它同时解决了以下的两个问题：

1.1 减少开发成本

无论是 Android 还是 iOS 平台，React Native 都使用 JavaScript 作为主要的开发语言，同时它也使用了 React 来简化和加速开发流程。由于 Android 与 iOS 中的系统 UI 组件大部分都相似（如文字、图片、列表等），基础功能（如本地存储、动画、通知）也基本类似，因此 React Native 将这些组件统一抽象为一个 Module，开发者不用再关心到底当前编写的 Module 运行在哪个平台上，只需要关注本身的业务逻辑即可。

而对于 Android 与 iOS 所各自特有的概念，React Native 进行单独定义，如 Android 中的 Toolbar、iOS 中的 TabBar 等等。这样开发者就可以针对不同平台的特性，在现有代码基础上稍作修改以适配不同的移动操作系统。Android 软件的设计与其他系统中软件设计基本类似[6]，可能在测试方法、测试用例的编写上有所不同[7]，但是毕竟开发流程并不同意。

Facebook 在提出 React Native 的时候提出“Learn once，write anywhere”，即不同的平台代码不会完全相同，但是会很相似。即也可以保证一套开发人员即可完成多平台业务场景的开发与实现，相较于传统的开发方式有很大改观。

1.2 提供一致的用户体验

一般来讲，在各自平台使用原生的 API 开发的应用是运行效率最高的，即用户体验最好的。而基于 Web 形式的应用既需要加载时间，又需要有较强的处理器进行 DOM 渲染，因此在不同平台、不同设备上的体验非常不一致。虽然说 JQuery 等开源方案经过优化可以达到比较好的效果，但是离原生应用差距还是较大。[8]

举例来说 PhoneGap[9]属于较为成熟的基于 Web 的跨平台解决方案，它在 iOS 上的流畅度和加载时间要远远好于 Android，或 Android 中只能处理一些简单的静态逻辑。如果仅仅使用 PhoneGap 为 iOS 开发，就失去了跨平台解决方案的意义。

React Native 虽然使用 JavaScript 作为开发语言，但是它并不将界面 UI 渲染到 Web 上，而是使用 JavaScript 解释器，将 JS 代码解释为原生的组件再渲染到界面上。因此，从用户体验上来讲，React Native 的渲染效率应该与原生应用是一致的。它与 Web 类的开发框架的不同之处在下一章进行详细讲解。

2 React Native 运行原理

由于 Android 版本的 React Native 刚处于测试阶段，因此这里选取 iOS 作为分析对象，对不同平台来说，它们的实现原理基本类似。

2.1 JS 与原生模块相互调用

在 iOS 中，React Native 使用自带的 JavaScriptCore 作为 JS 的解析引擎，但并没有用到 JavaScriptCore 提供的一些可以让 JS 与 OC 互调的特性，而是自己实现了一套机制，这套机制可以通用于所有 JS 引擎上，这么做应该也是考虑到了项目的灵活性与 iOS 安全机制的问题。

实际上普通的 JS 与原生代码的调用非常简单[10]，无论是 Android 还是 iOS，原生代码可以向 JS 传递消息，如 iOS 中 webview 提供的 stringByEvaluatingJavaScriptFromString 方法可以直接执行一段 JS 脚本，并且可以获取它的返回值。这个返回值其实就相当于 JS 向原生代码传递消息。React Native 就是以此为基础，通过模块配置表和响应流程实现了原生代码与 JS 代码的无缝衔接调用。

2.2 模块配置表

在 React Native 初始化的时候，为了使 JS 知道可调用的原生模块 Module 方法与命名，需要将一份 Module 的配置表传递给 JS，配置表里包含了所有的模块和模块中的方法信息。

React Native 框架中提供了一个叫做 RCTBridge Module 的接口。在 React Native 初始化的时候，可以通过 runtime 接口 objc_getClassList 获取所有的类，然后逐个判断是否实现了 RCTBridgeModule 接口，就可以找到所有的模块类了。一个模块中有很多方法，一些可以暴露给 JS 调用，另一些则不需要。需要暴露的方法使用 RCT_EXPORT_METHOD 宏定义包裹，其实是将方法名在编译时重新定义为有固定前缀的方法，方便查找并记录。

因此在原生代码中定义好了接口与函数，React Native 就会根据运行时环境获取所有的模块与其方法

生成模块配置表。这个配置表在原生代码与 JS 代码之间都维持一份相同的配置，便于原生代码与 JS 代码相互调用。

2.3 事件传递响应流程

React Native 的主要核心是自己实现了一套 JS 与原生代码相互调用的机制，这也是与其他基于 Web 的跨平台方案不同的地方。使用 JS 写好业务代码后，由 React Native 进行解释，偏向效率的逻辑执行、UI 渲染都会在原生阶段进行处理，因此 React Native 的执行效率要比其他的跨平台方案高很多。

JS 与原生代码的调用流程比较负责，如下所示：

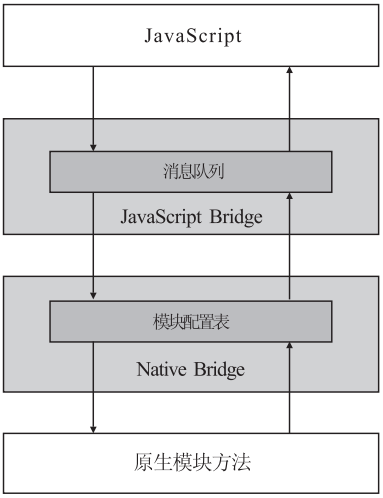

图 1 JavaScript 与 Native 调用流程

1、JS 端调用原生模块中的某个方法

2、将调用分解为模块名、方法名、参数，传递给 MessageQueue 进行处理，生成 callbackId

3、将模块名、方法名、参数对应模块配置表，转换成模块 id、方法 id、参数 id

4、原生模块根据各个 id 确定被调用的方法与参数

5、执行原生代码逻辑，将执行结果进行回调

6、JS 端根据 callbackId 接收执行结果

其中需要特别注意的是，JS 并不会主动传递数据给 OC。在调用 OC 方法时，会把模块 id、方法 id 放入到一个队列中。等原生代码过来调用 JS 的任意方法时，将这个队列返回再执行这个队列中需要执行的方法。所以 React Native 是一种基于事件响应机制的框架，在没有操作时不会进行无效的循环执行。

3 代码压缩与动态加载

3.1 动态加载的概念

在移动平台安装或更新软件时，通常使用 apk 或 ipa 文件进行安装，也就是所称的安装包。这个安装包中包含了应用全部功能的实现。然而通常情况下，用户只有下载完整的安装包才能使用最新版的软件系统。动态加载的概念则是只下载更新部分的特定代码（文件形式），覆盖到原来的代码空间，这个概念与我们在 Windows 平台上所说的补丁概念类似。

在移动平台的背景下，由于用户的网络环境可能更差、存储空间更小，因此动态加载就非常适用于移动平台软件的更新与需求实现。另外一方面，由于移动互联网发展迅速、软件迭代频繁，动态加载可以在不下载完整安装包的情况下，让用户没有感知的完成应用的更新，更有利于修复线上 bug 与需求的快速实现。

React Native 就完全支持动态加载的这个特性，当开发人员编写好新的功能模块时，将业务代码与所需要的资源（Assets）打包下发到软件中，软件再动态加载业务代码，实现动态加载。主要流程如下图所示：

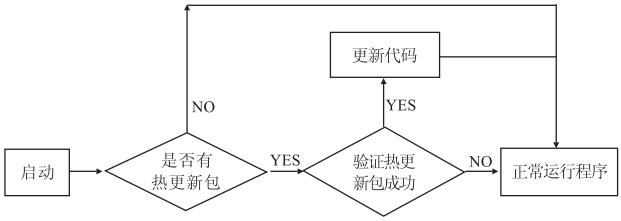

图 2 热更新包加载流程

需要注意的是，在下载更新包之后，需要对代码的版本、安全性进行验证，防止打错版本或第三方对软件进行篡改劫持。

3.2 简化 JS 代码结构

在 React Native 当中，可以对 JS 业务代码进行打包处理，即 JS 代码既可以在线访问也可以离线访问。使用如下命令可以将 JS 业务代码进行打包：

react-native Bundle – minify

这样就可以将 JS 业务代码打包成一个 bunlde 文件，在 app 运行时指定这个文件的本地地址（或在线地址）就可以对其进行访问并解释执行。但是有一个非常明显的缺陷是，使用 React Native 提供的 bunlde 命令打包出的 JS 文件代码量非常的大，至少有 5 万行以上。React Native 应用加载的流程如下图所示：

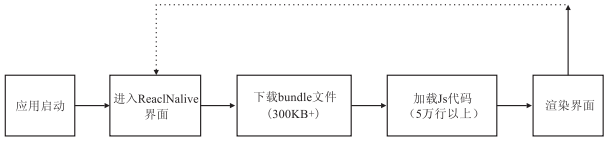

图 3 ReactNative 加载流程

从图中可以看到，第三步与第四步中是较为耗时

的两个过程。没有经过处理的 bunlde 文件体积通常在 300KB 以上，如果使用 2G 网络传输则需要至少十秒以上，这对于用户体验有非常大的影响。另外，过多的代码量对移动的设备的性能也会造成影响，5 万行代码的执行时间通常也在 200 到 300 毫秒之间，虽然说时间并不多，但是用户可以明显感受到加载界面时的卡顿现象。

因此在本研究当中针对这两个非常耗时的环节进行优化，提出了一种分离基础 JS 与业务 JS 的方法，可以大大缩小代码的下载与加载时间。如下图所示：

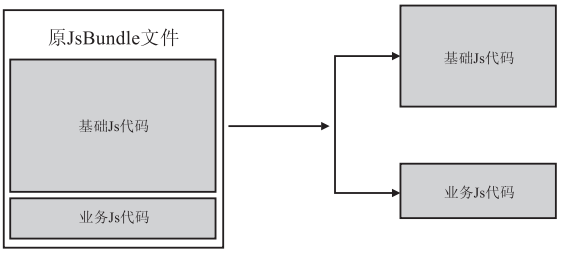

图 4　JS 代码简化与分离

如上图所示，原有的 Bundle 文件中，同时夹杂了基础 JS 代码与业务 JS 代码。基础 JS 代码指的是业务代码运行时所依赖的基础环境。我们为不同模块打包出的的 Bundle 文件中都会包含基础 JS 代码，因此如果软件中同时有多个 Bundle 文件时，会造成空间的浪费。经过验证基础 JS 代码大概在 5 万行左右，而业务 JS 根据业务复杂程度而定，一般不超过 10KB，这样就非常有利于业务 JS 在网络中进行传递与动态加载。

3.3　动态加载与实验验证

在分离了 JS 代码之后，就可以针对具体的业务场景进行优化。由于基础 JS 文件较大，不适合在网络中进行传输，因此可以将其缓存在本地。而每次启动应用进行更新时，只需要下载体积非常小的业务 JS。流程如下图所示：

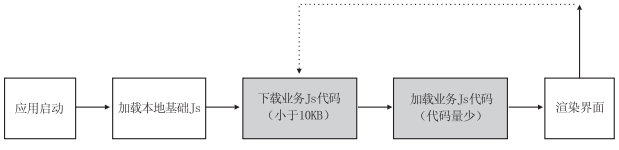

图 5　简化后的加载流程

应用启动后，加载本地的基础 JS 代码，且在不同的业务场景切换时不再次加载。即基础 JS 在各个业务场景间进行共享。其次从网络中下载业务 JS 代码，由于单个业务场景逻辑本身实现并不需要非常大的代码量，因此一般业务 JS 打包并压缩之后可以保持在 5KB 左右，使用 2G 网络进行下载时仅仅需要 1 秒就可以完成。同时由于不再需要加载基础 JS，渲染业务场景的时间也会更少，经过实验验证仅需 10 到 50 毫秒即可渲染完毕。相较于之前的 200 毫秒有较大的提升。如下表所示：

表 1　优化前后性能比较

	代码大小	网络传输时间（2G）	渲染时间
优化前	>300KB	>10s	300ms
优化后	<10KB	<1s	50ms

4　结论

React Native 不同于其他的跨平台开发框架，它同时解决了开发成本和运行效率的两个关键问题。在此基础之上本论文优化了其工程代码结构，将基础 JS 与业务 JS 相互隔离，两部分可独立更新与迭代替换。其次在使用 React Native 进行动态加载时，仅需加载业务 JS 即可，大幅减少了在移动网络中传输的数据大小，加快了界面的载入速度，同时也明显提升了应用的用户体验。

参考文献

[1] 武佳佳，王建忠.基于 HTML5 实现智能手机跨平台应用开发[J].软件导刊，2013，02：66-67.

[2] 唐灿.下一代 Web 界面前端技术综述[J].重庆工商大学学报（自然科学版），2009，04：351-355.

[3] 张景安，张杰，卫泽丹.Android 移动端浏览器的设计与开发[J].软件，2014，35（8）：7-10.

[4] Facebook.React Native[OL].[2015-12-19] http：//facebook.github.io/react-native.

[5] Facebook.React JS[OL].[2014-07-23].http：//facebook.github.io/react/.

[6] 黄文雄.面向 Android 应用的用户行为分析方法[J].软件，2014，35（12）：83-87.

[7] 刘璐.Android 智能终端功能测试方法设计与系统实现[J].软件，2014，35（12）：79-82.

[8]付予.一种基于 jQuery 的跨模块通信机制的改进方案[J].软件，2014，35（12）：42-44.

[9] 杨叶，陈琳，董启标.基于 PhoneGap 的跨平台移动学习资源设计与开发探究[J].现代教育技术，2014，2：100-107.

[10] 李张永，陈和平，顾进广.跨平台移动 Web 开发框架与数据交互方法[J].计算机工程与设计，2014，05：1828-1832.