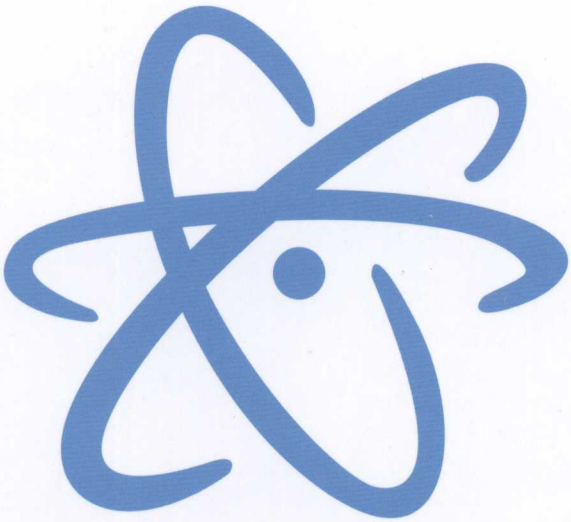
React Native 移动开发实战

向治洪 著

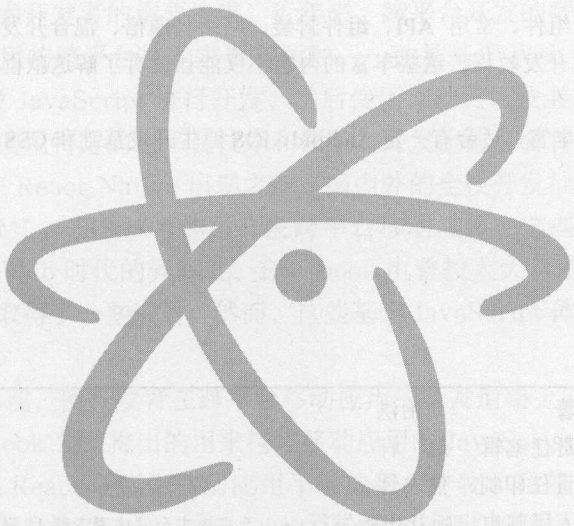


React Native 框架超全面解析！
手把手教你打造高品质移动用户体验！

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

React Native
移动开发实战

向治洪 著



人民邮电出版社
北 京

图书在版编目（CIP）数据

React Native移动开发实战 / 向治洪著. -- 北京 : 人民邮电出版社, 2018.1
ISBN 978-7-115-47096-6

Ⅰ. ①R… Ⅱ. ①向… Ⅲ. ①移动终端－应用程序－程序设计 Ⅳ. ①TN929.53

中国版本图书馆CIP数据核字(2017)第276301号

内 容 提 要

本书全面详尽地介绍了 React Native 框架的方方面面，内容涵盖 React Native 基础知识、环境搭建与调试、开发基础、常用组件、常用 API、组件封装、网络与通信、混合开发、热更新与打包部署，以及两个实际案例的完整开发教程。这些丰富的内容不仅能让读者了解这款框架中涉及的各类概念，还能指导读者开发实践。

本书语言简洁，内容丰富，适合有一定 Android、iOS 原生开发基础和 CSS 基础的移动开发工程师学习。

◆ 著　　向治洪
责任编辑　赵　轩
责任印制　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京市艺辉印刷有限公司印刷

◆ 开本：800×1000　1/16
印张：21
字数：496 千字　　2018 年 1 月第 1 版
印数：1－3 000 册　　2018 年 1 月北京第 1 次印刷

定价：69.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

前言

PREFACE

近年来，随着移动互联网产业的持续快速发展，以及智能手机、智能电视等智能终端设备的普及，移动互联网应用获得了爆炸式增长。面对未来广阔的市场，运营商、互联网企业、设备生产商等产业巨头纷纷构建了移动互联网生态链，其中苹果公司和 Google 创造的移动互联网应用商业模式，激发了广大开发者开发移动互联网应用的热情。

众所周知，原生（Nutive）应用因其性能优秀、体验较好而获得了广大用户和开发者的欢迎。然而，原生应用开发周期长、支持设备有限等问题也困扰着开发者和商户，因而，跨平台移动应用开发成为技术开发者的重要追求。近年来，跨平台开发的呼声越来越高，已成为一种趋势。目前，移动应用的跨平台技术主要有两种。一种是基于 Web 的移动开发技术，只需要使用标准的 HTML 及 JavaScript 进行开发，然后使用移动终端安装的浏览器，即可实现应用的跨平台；另一种是使用特定的跨平台技术和框架，开发出能在各种主流移动操作系统上运行的 APP 应用程序。在 React Native 出现之前，国内外的全栈开发社区都在坚持不懈地寻求使用 JavaScript 和 HTML、CSS 技术体系开发跨平台移动应用，这些技术被统称为 H5 技术（HTML5 技术）。作为 Web 时代的弄潮儿，Facebook 也曾投入大量的人力物力，在移动 H5 技术上攻坚克难，虽然取得了一些进展，然而，这些基于 JavaScript 体系开发的移动应用始终达不到理想的效果。

React Native 的出现，使开发者在跨平台移动应用的开发道路上向前迈了一大步。React Native 是一款由 Facebook 公司推出的用来构建移动应用的 JavaScript 框架，是 Facebook 早先开源的界面渲染框架 React 在原生移动应用平台的衍生产物，目前支持 iOS 和 Android 两大平台。React Native 倡导的“Learn once, write anywhere”（仅需学习一次，编写任何平台）也赢得了广大开发人员的青睐。虽然，新框架的引入不可避免地增加了学习成本，但是，相对于其他的跨平台技术而言，React Native 的学习成本还是比较低的。截至 2017 年 7 月，在 GitHub 上 React Native 已获得了 52000 多个 star，成为时下最受欢迎的跨平台移动应用开发框架之一。

在技术实现上，React Native 抛弃了传统的浏览器加载思路，转而采用曲线调用原生 API 的思路来渲染界面，从而获得了媲美原生应用的体验。React Native 具体实现思路如下：应用启动后会从服务器下载最新的 JSBundle 文件，然后通过本地的 JavascriptCore 引擎对 JS（Javascript 缩写）文件进行解析，并利用 Bridge 映射到对应的原生 API 上，进而调用原生方法和 UI 组件来渲染界面。在语法上，React Native 使用 JSX 来替代常规的 JavaScript，这

是一种很像 XML 的 JavaScript 语法扩展。因此，熟悉 JavaScript 类库的 Web 开发者可以使用 React Native 轻松地开发出移动应用。由于使用 JSX 编写的大部分代码可以实现平台间共享，因此，采用 React Native 开发可以大幅减少跨平台移动应用开发的工作量。同时，React Native 框架采用模块化结构，使应用版本的更新迭代也异常简单。当然，React Native 也不是完美无缺的，但瑕不掩瑜，随着它的日趋成熟，React Native 势必会成为跨平台移动应用开发的主流技术。

本书适当地介绍了一些原理性的概念，但并不深究，同时本书提供的不少案例，也将带领你快速地进入 React Native 的世界。雄关漫道真如铁，而今迈步从头越。相信通过学习本书，你一定会有所收获。

作者

目录

CONTENTS

目录

第1章 React Native入门

1.1 React Native基本知识

React Native 基于 React 框架而设计，因此，了解 React 有助于我们更好地理解 React Native。

1.1.1 React简介

React 是由 Facebook 推出的前端开发框架，其本身作为 MVC 模式中的 View 层来构建 UI，也可以以插件的形式应用到 Web 应用程序的非 UI 部分构建中，轻松实现与其他 JS 框架的整合。同时，React 通过对虚拟 DOM 的操作来控制真实的 DOM，从而得到页面的局部更新，提高了 GPU 渲染的性能，而 React 提出的模块化开发思路也大大提高了代码的可维护性。

React 的官方地址是 https://github.com/facebook/react。截至 2017 年 4 月，React 获得了超过 62K 的 star 和 11K 的 fork，这说明 React 得到了技术人员的普遍支持，正是由于这些原因，React.js 和 Vue.js、Angular.js 成为了当今最流行的三大前端框架。

讲到 React，就不得不提到组件（Component）的概念，它是 React 最基础的部分，其功能相当于 AngularJS 里面的 Directive，或是其他 JS 框架里面的 Widgets 或 Modules。Component 可以认为是由 HTML、CSS、JavaScript 和一些内部数据组合而成的模块，当然 Component 也可以由很多其他的 Component 组建而成。不同的 Component 既可以用纯 JavaScript 定义，也可以用特有的 JavaScript 语法 JSX 创建而成。采用 React 进行项目开发，能够获得以下优势。

虚拟DOM（Virtual DOM）

传统的 Web 应用开发，一般都是通过直接操作真实 DOM 来进行更新操作的，如图 1-1 所示，但对 DOM 进行操作通常是比较昂贵的。而 React 为了尽可能减少对真实 DOM 的操作，采用了一种强大的方式来更新 DOM，代替直接的 DOM 操作，这就是 Virtual DOM，一个轻量级的虚拟 DOM。

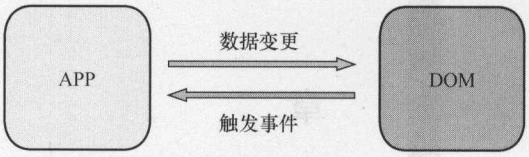


图1-1 传统的Web应用结构图

虚拟 DOM 其实是 React 抽象出一个对象，通过这个 Virtual DOM 可以更新真实的 DOM，由这个 Virtual DOM 管理真实 DOM 的更新，如图 1-2 所示。简单来说，React 在每次需要渲染时，会先比较当前 DOM 内容和待渲染内容的差异，然后再决定如何最优地更新 DOM，这个过程被称为 reconciliation。

图1-2 React Web应用结构图

除了性能方面的考虑，React 引入虚拟 DOM 更重要的意义在于提供了一种新的开发方式来开发服务端应用、Web 应用和手机端应用，如图 1-3 所示。

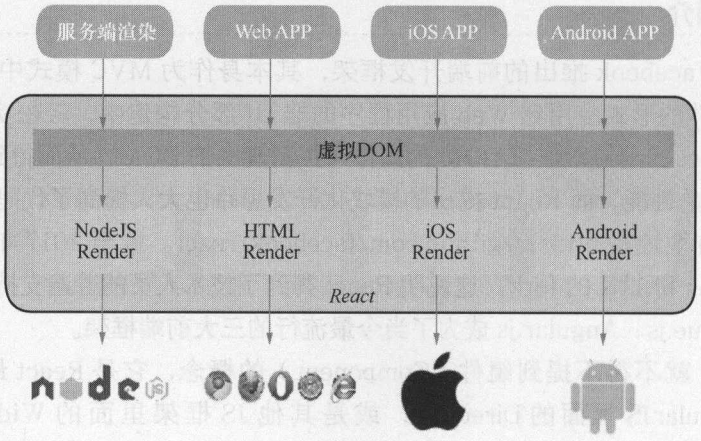


图1-3 虚拟DOM结构图

Components组件

虚拟 DOM（virtual-dom）不仅带来了简单的 UI 开发逻辑，同时也带来了组件化开发的

思想。所谓组件，即自己封装的具有独立功能的 UI 部件。React 推荐以组件的方式去构成视图，并建议将功能相对独立的模块抽象为组件。例如，Facebook 的 instagram.com 网站都采用 React 来开发，整个页面就是一个大的组件。

对于 React 而言，界面被分成不同的组件，每个组件都相对独立。在 React 开发中，整个界面可以看成是由大小组件构成，每个组件实现自己的逻辑部分即可，彼此独立，如图 1-4 所示。

采用组件化开发，往往具有以下特点：

- 可组合（Composeable）：一个组件易于和其他组件一起使用，或者嵌套在另一个组件内部。如果在一个组件内部创建了另一个组件，那么父组件拥有它创建的子组件，通过这个特性，一个复杂的UI可以拆分成多个简单的UI组件。
- 可重用（Reusable）：每个组件都可以独立出来，被使用在其他相似的UI场景中。
- 可维护（Maintainable）：每个小的组件仅仅包含自身的逻辑，更容易被理解和维护。
- 可测试（Testable）：每个组件都是独立的，那么对于各个组件分开测试显然要比整个界面容易得多。

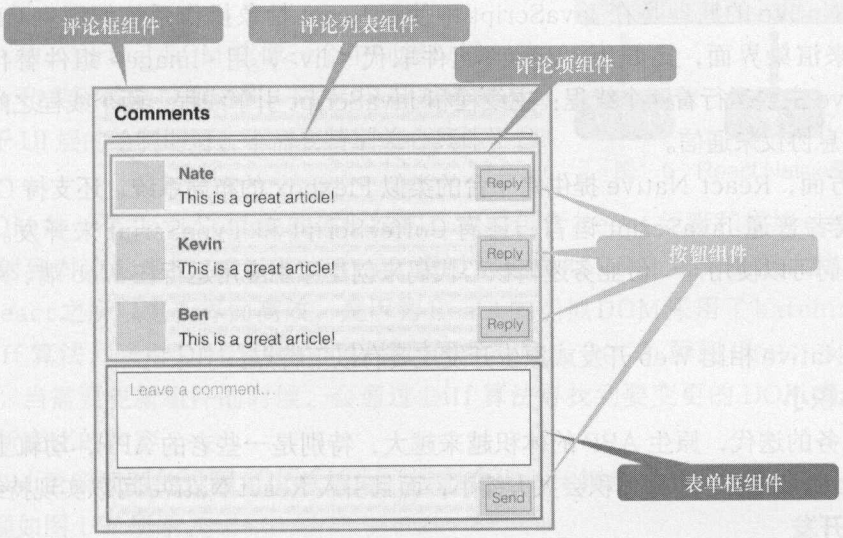


图1-4　Components组件示意图

数据流（Data Flow）

React 采用单向的数据流，即从父节点到子节点的传递，因此更加灵活便捷，也提高了代码的可控性。React 单向数据流可以总结为以下流程：Action → Dispatcher → Stroe → View，如图 1-5 所示。

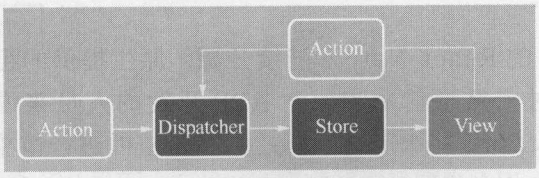


图1-5　React单向数据流流程图

JSX语法

JSX 是 React 的核心组成部分，React 使用 JSX 来替代常规的 JavaScript。它使用 XML 标记的方式去直接声明界面，目的是通过各种不同的编译器将这些标记编译成标准的 JS 语言。使用 JSX 语法后，可以让组件的结构和组件之间的关系看上去更加清晰并且执行效率更高。

1.1.2 React Native简介

React Native（简称 RN）是 Facebook 于 2015 年 4 月开源的跨平台移动应用开发框架，是 Facebook 早先开源的 UI 框架 React 在原生移动应用平台的衍生产物，目前支持 iOS 和 Android 两大平台。

React Native 可以基于目前大热的开源 JavaScript 库 ReactJS 来开发 iOS 和 Android 移动应用，因为往往只需要开发一套代码就可以满足 iOS 和 Android，正如 Facebook 说的“Learn once，write anywhere”（仅需学习一次，编写任何平台），由于基于 Web 技术，React Natie 开发起来可以像在浏览器那样随改即所见。

React Native 的原理是在 JavaScript 中使用 React 抽象操作系统的原生 UI 组件，代替 DOM 元素来渲染界面，比如用 <View> 组件取代 ，用 <Image> 组件替代 <img> 等。React Native 主要运行着两个线程：主线程和 JavaScript 引擎线程，两个线程之间通过批量化的 async 消息协议来通信。

在 UI 方面，React Native 提供跨平台的类似 Flexbox 的布局系统，还支持 CSS 子集。可以用 JSX 或者普通 JavaScript 语言，还有 CoffeeScript 和 TypeScript 来开发。运用 React Native，我们可以使用同一份业务逻辑核心代码来创建原生应用运行在 Web 端、Android 端和 iOS 端。

React Native 相比 Web 开发或原生开发主要有以下特点：

APP占用体积小

随着业务的迭代，原生 APP 的体积越来越大，特别是一些老的 APP，动辄上百兆，而采用 React Native 之后，占用体积会大大缩小，而且引入 React Native 可以实现持续开发。

实现跨平台开发

基于 Web 技术（HTML5/JavaScript）构建的移动应用速度快，开发周期短，但是体验较差，响应不及时；而使用原生开发，周期长，项目风险不可控。如何提高开发效率，节约人力成本。成为各大公司考虑的问题，而 React Native 的出现解决了上面的问题，只需要开发一套代码，便可以同时部署到 Android 和 iOS 两个移动平台上。

相对成熟的技术

随着 Android/iOS 的 React Native 陆续开源，原生提供的组件和 API 相对丰富，且实现技术基本一致，对于熟悉前端和原生 APP 开发的人员来说很容易上手。而 React Native 通过 JavaScriptCore 将 JS 转换为原生 APP 组件进行渲染，其用户体验也可媲美原生 APP。

支持动态更新

在原生 APP 开发中，Android 平台可以通过插件化实现热更新。在 iOS 平台上，热更新策略是严令禁止的（如 JSPatch/wax/rollout 等技术），而采用 React Native 技术完全可以满足要求，而又不触碰苹果的底线。

1.1.3 React Native工作原理

使用 JavaScript 开发移动 APP 的想法来源于最近几年市场对于移动应用需求的增长，为了快速开发一款可以使用，而体验又不是那么糟糕的 APP，很多公司投入大量人力开发跨平台应用。而在这些公司当中，Facebook 无疑是做得最好、最成功的。

为了更好地理解 React Native，我们需要对 React Native 的工作原理和整体架构有一个了解，如图 1-6 所示。

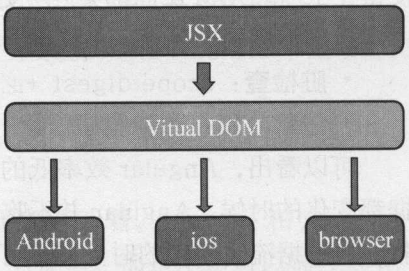


图1-6 React Native整体框架图

如图，React Naitve 框架分为 3 层，分别为：JSX 环境层、虚拟 DOM 层、具体的平台层。这里面最重要的就是虚拟 DOM 层。

在 React 中，Virtual DOM 就像一个中间虚拟层，位于 JavaScript 和实际渲染页面之间。对于 JS 开发者来说，只需要专注于 UI 层的绘制即可，不需要特别关心具体平台的实现。

虚拟 DOM 是一个 JavaScript 的树形结构，主要包含 React 元素和模块。组件的 DOM 结构就是映射到对应的虚拟 DOM 上，React 通过渲染虚拟 DOM 到浏览器，使得用户界面得以显示。React 之所以更新界面高效，是因为 React 的虚拟 DOM 采用了 batching（批处理）和高效的 Diff 算法，采用 Diff 算法，可以将时间复杂度从 O(n^3) 降到 O(n)，从而提高界面构建的性能。当需要更新组件的时候，会通过 Diff 算法寻找到要变更的 DOM 节点，然后通知浏览器更新变化的内容。

虚拟 DOM 更新视图的过程可以总结为：状态变化→计算差异（Diff 算法）→界面渲染。其渲染的原理如图 1-7 所示。

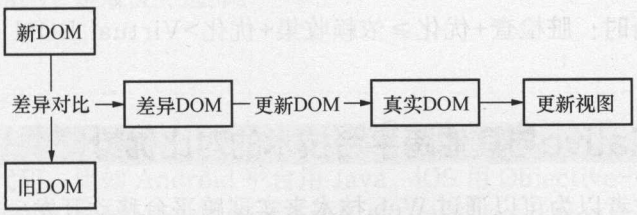


图1-7 虚拟DOM更新原理图

在界面渲染过程中，React Native 针对不同的平台调用原生 API 去渲染界面，例如 iOS 平台调用其原生的 API 去渲染 iOS 界面，Android 平台调用其原生 API 去渲染 Android 界面，而

不是直接渲染到浏览器的 DOM 上，这使得 React Native 不同于基于 Web 视图的跨平台应用开发方案，因而，采用 React Native 开发的 APP，体验更加接近原生 APP。

虚拟DOM和MVVM的对比

虚拟 DOM 只是 MVVM 框架的一种实现方案，二者没有好坏之分。在流行的前端框架中，除了 React 采用虚拟 DOM 之外，其他 MVVM 系框架，如 Angular、Vue、Avalon，采用的都是数据绑定。

何为数据绑定？简单来说，就是通过观察 Directive/Binding 对象数据变化，并保留对实际 DOM 元素的引用，当有数据变化时进行对应的操作。React 检查是 DOM 结构层面的，而 MVVM 的检查则是数据层面的。MVVM 的性能检测也根据检测层面的不同而有所不同：Angular 的脏检查使得任何变动都会产生固定的更新的代价；而 Vue/Avalon 采用的依赖收集，使得在 JS 和 DOM 层面都会产生更新。

上面提到两个概念——脏检查和依赖收集。

- 脏检查：scope digest +必要DOM更新
- 依赖收集：重新收集依赖+必要DOM更新

可以看出，Angular 效率低的地方在于任何小变动都会引起界面的重绘，但是，当所有数据都变化的时候，Angular 并不吃亏。依赖收集在初始化和数据变化的时候都需要重新收集依赖，在数据流比较小的时候几乎可以忽略，但在数据量比较大的时候就会产生一定的消耗。相比之下，React 的变动检查则是 DOM 结构层面的，即使是全新的数据，只要渲染结果没有变化，也不需要重新绘制。

Angular 和 Vue 都提供了重绘的优化机制，即有效地复用实例和 DOM 元素。在优化的版本中，Angular 和 Vue 采用了 track by $index 技术后比 React 的效率更高。

所以在框架选择和技术性能分析的时候，要分清楚初始渲染、小量数据更新、大量数据更新这些不同的场合，以及 DOM、脏检查 MVVM、数据收集 MVVM 在不同场合各自的表现和优缺点，具体表现和区别如下。

- 初始渲染阶段：Virtual DOM >脏检查≥依赖收集
- 小量数据更新时：依赖收集>Virtual DOM +优化>脏检查（无法优化）> Virtual DOM无优化
- 大量数据更新时：脏检查+优化≥依赖收集+优化>Virtual DOM（无优化）> MVVM无优化

1.2 React Native与其他跨平台技术的对比优势

曾经大部分开发者以为可以通过 Web 技术来实现跨平台移动开发，却因为性能限制或其他问题而放弃，最终，不得不针对多个平台开发多个版本，这违背了跨平台开发的初衷。而 React Native 的出现让跨平台移动端开发再次回到人们的视野中，而它提倡的“Learn once, write anywhere”也赢得了广大开发人员的青睐。相比传统的 H5 技术，React Native 获得了

更加接近原生应用的体验。

为了方便理解，笔者将跨平台技术分为四大流派。

- Web流：也被称为Hybrid技术，它基于Web相关技术来实现界面及功能。
- 代码转换流：将某个语言转成Objective-C、Java或C#，然后使用不同平台下的官方工具来开发。
- 编译流：将某个语言编译为二进制文件，生成动态库或打包成apk/ipa/xap文件。
- 虚拟机流：通过将某个语言的虚拟机移植到不同的平台上来运行。

1.2.1 Web流

Web流，如大家熟知的PhoneGap/Cordova等技术，它将原生的接口封装后暴露给JavaScript，然后通过系统自带的WebView运行，也可以使自己内嵌Chrome内核。

Web流缺点是性能差、渲染速度慢。说它Web性能差，主要说的是在Android下比较差，在iOS下已经很流畅了。

性能差的主要原因是，在Android和iOS的早期设备中，由于没有实现GPU加速，所以会造成每次重绘界面的卡顿。

而造成渲染慢的第二个原因是：CSS过于复杂。因为从实现原理上看，Chrome和Android View并没有本质上的差别，但过于复杂的CSS会加重GPU的负担。那是不是可以通过简化CSS来解决呢？实际上还真有人进行了这种尝试，比如著名的Famo.us，其最大的特色就是不使用CSS，只能使用固定的几种布局方法，完全依靠JavaScript来写界面，它能有效避免低效的CSS代码，从而提升机器性能。

造成绘制缓慢的第三个原因是，业务需求的复杂，比如超长的ListView商品展示。因为DOM是一个很上层的API，使得JavaScript无法做到像Native那样细粒度地控制内存及线程，所以难以进行优化，特别是在硬件较差的机器上。

上面三个问题现在都不好解决。其实除了性能之外，Web流更严重的问题是功能缺失。比如iOS 8就新增4000多个API，而Web标准需要漫长的编写和评审过程，而等到Web审核通过，即便是Cordova这样的优秀的框架，或者自己封装也是忙不过来的。所以为了更好地使用原生系统新功能，Native是最快的选择。

1.2.2 代码转换流

不同平台下的官方语言不一样，并且平台对官方语言的支持最好，这就导致对于同样的逻辑，我们需要写多套代码。比如Android平台用Java，iOS用Objective-C或者Swift。于是就有人想到了通过代码转换的方式来减少重复的工作量，这就是代码转换流。

这种方式虽然听起来不是很靠谱，但它的成本和风险都是最小的，因为代码转换后就可以用官方提供的各种工具了，和普通开发区别不大，而且转换后，利用原生的优势，可以减少兼容性问题。

目前存在以下几种代码转换方式。

将Java转成Objective-C

2objc 是一款能将 Java 代码转成 Objective-C 的工具，据说 Google 内部就是使用它来降低跨平台开发成本的，比如 Google Inbox 项目就号称通过它共用了 70% 的代码，效果很显著。有了 2objc，我们就可以先开发 Android 版本，然后再开发 iOS 版本。

将Objective-C转成Java

MyAPPConverter 是一款将 Objective-C 代码转换成 Java 代码的工具，比起前面的 2objc，MyAPPConverter 还打算将 UI 部分也包含进来，从它已转换的列表中可以看到还有 UIKit、CoreGraphics 等组件，使得有些应用可以不改代码就能转换成功。

XMLVM

除了上面提到的源码到源码的转换，在代码转换流中，还有 XMLVM 这种与众不同的转换方式，它首先将字节码转成一种基于 XML 的中间格式，然后再通过 XSL 来生成不同语言，目前支持生成 C、Objective-C、JavaScript、C#、Python 和 Java。

虽然基于中间字节码可以支持多语言，但是这种方式也有一些问题，例如生成代码不可读，因为很多语言中的语法会在字节码中被抹掉，并且是不可逆的，所以不利于代码的调试和发现问题。

综上所述，虽然代码转换这种方式风险小，但对于很多小 APP 来说其实共享不了多少代码，因为这类应用大多数围绕业务来开发的，大部分代码都和业务逻辑耦合，所以公共部分不多，其意义不大。

1.2.3 编译流

编译流比代码转换流的代码转换更进一步，它直接将某个语言编译为普通平台下能够识别的二进制文件。采用这种方式主要有以下特点。

优点

- 可以重用一些实现很复杂的代码（比如之前用C++实现的游戏引擎，重写一遍的成本太高）。
- 编译后的代码反编译困难，安全性更好。

缺点

- 转换过于复杂，并且后期定位和修改成本会很高。
- 编译后体积太大，尤其是支持ARMv8和x86等CPU架构的时候。

常用的编译流方案如下所示。

C++方案

因为目前 Android、iOS 和 Windows Phone 都提供了对 C++ 开发的支持。特别是 C++ 在实现非界面部分，性能是非常高的。而如果使用 C++ 实现非界面部分，还是比较有挑战的。这主要是因为 Android 程序的界面绝大部分是 Java 编写的，而在 iOS 和 Windows Phone 平台下

可以分别使用 C++ 的超集 Objective-C 和 C++/C# 来开发。要解决使用 C++ 开发 Android 应用程序界面的问题，目前主要有两种方案。

- 通过JNI调用系统提供的Java方法。
- 自己实现UI部分。

第一种方式虽然可行，但是代码冗余高，实现过于复杂。那第二种方式呢，比如 JUCE 和 Qt 就是用代码实现的。不过在 Qt 的方案中，Android 5 版本或更高版本环境下，很多效果都没法实现，比如按钮没有涟漪效果。根本原因在于它是通过 Qt QUIck Controls 的自定义样式来模拟的，而不是使用系统 UI 组件，因此它享受不到系统升级自动带来的界面优化。

当然我们可以使用 OpenGL 来绘制界面，因为 EGL+OpenGL 本身就是跨平台的。并且目前大多数跨平台游戏底层都是这么做的。

既然可以基于 OpenGL 来开发跨平台游戏，那么，是否能用它来进行界面开发呢？当然是可行的，而且 Android 4 的界面就是基于 OpenGL 的，不过它并不是只用 OpenGL 的 API，那样是不现实的，因为 OpenGL API 最初设计并不是为了实现 2D 界面，所以连画个圆形都没有直接的方法，因此 Android 4 中是通过 Skia 将路径转换为位置数组或纹理，然后再交给 OpenGL 从而完成界面渲染的。

然而直接使用 OpenGL 绘制界面，不仅实现的代价大，而且目前支持的平台少。因此对于大多数应用来说自己实现界面是很不划算的。

Xamarin

Xamarin 是从 Mono 发展而来，它用 C# 来开发 Android 及 iOS 应用，因为相关工具及文档都挺健全，因而发展得还不错。在 UI 界面方面，它可以通过调用系统 API 来使用系统内置的界面组件，或者基于 Xamarin.Forms 开发定制要求不高的跨平台 UI。

从实现的方式来讲，iOS 下是以 AOT 的方式编译为二进制文件的；而在 Android 平台上是通过内嵌的 Mono 虚拟机来实现，所以 Xamarin 是跨平台开发的不错选择。

对于熟悉 C# 的团队来说，Xamarin 是一个很不错的方案，但这种方案最大的问题就是相关资料不足，遇到问题很可能搜不到解决方案，并且当前第三方库太少，加之 Xamarin 本身有些 bug，所以让我们静待 Xamarin 做得更好吧。

Go

Go 做为后端服务开发语言，专门针对多处理器系统应用程序的编程进行了优化，使用 Go 编译的程序可以媲美 C 或 C++ 程序的速度，而且更加安全、支持并行进程。Go 从 1.4 版本开始支持开发 Android 应用（1.5 版本支持 iOS）。虽然能同时支持 Android 和 iOS，但是目前可用的 API 很少，Go 语言仍然专注于后端开发。

目前，Android 的 View 层完全是基于 Java 写的，要想用 Go 来完成界面的开发不可避免要调用 Java 代码，而在这方面 Go 还没有简便的实现方式，目前 Go 调用外部代码只能使用 Cgo，通过 Cgo 再调用 jni，这就不可避免地需要写很多的中间件。而且 Cgo 的实现本身就对性能有损失，除了各种无关函数的调用，它还会锁定一个 Go 的系统线程，这会影响