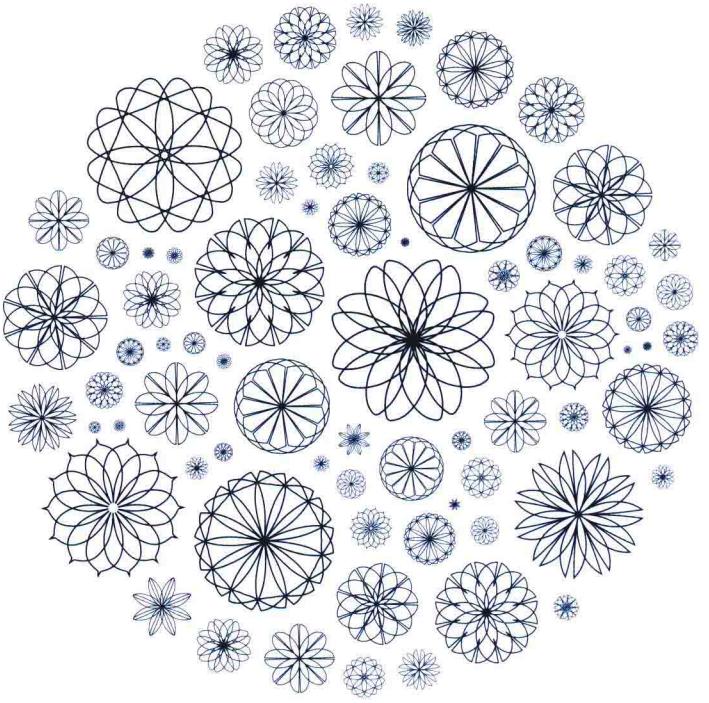
BLEEDING EDGE PRESS
Publishing at the speed of technology

TURING 图灵程序设计丛书

五位开发高手，三个企业应用示例，真实再现**React Native**开发场景
一次性开发跨平台应用、原生体验，开发**效率高**，满足前端开发**快速迭代**需求



Deconstructing React Native Apps

React Native
应用开发实例解析

[澳] Alexander McLeod [斯洛文尼亚] Pavlo Aksonov
[印] Arjun Komath [美] Atticus White [美] Isaac Madwed 著

林昊 译

中国工信出版集团

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

Deconstructing React Native Apps

React Native
应用开发实例解析

[澳] Alexander McLeod [斯洛文尼亚] Pavlo Aksonov
[印] Arjun Komath [美] Atticus White [美] Isaac Madwed 著

林昊 译

人民邮电出版社
北京

图书在版编目（CIP）数据

React Native应用开发实例解析 / （澳）亚历山大·麦克劳德（Alexander McLeod）等著 ; 林昊译. -- 北京：人民邮电出版社，2017.9
（图灵程序设计丛书）
ISBN 978-7-115-46714-0

Ⅰ. ①R… Ⅱ. ①亚… ②林… 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203246号

内 容 提 要

使用 React Native 可以轻松开发跨平台应用，并且无需等待 Apple、Google 或者 Amazon 的审核过程，就可以为自己的应用发布更新。本书主要从功能扩展和实际应用方面讲解 React Native，带领读者全面了解 React Native 的 API 和组件，并且阅读本书无需 React 开发背景。本书共五章，前两章介绍 React Native 的历史发展和基础知识，包括原生组件和第三方库；余下三章则分别介绍三个企业应用——Myagi、TinyRobot 和 Fixt，探讨了当今业界使用 React Native 的方式，以及生产环境下需要注意的问题和相应对策。

本书适合客户端开发人员、前端开发人员，以及所有对 React Native 感兴趣的程序员。

◆ 著　[澳] Alexander McLeod
[斯洛文尼亚] Pavlo Aksonov　[印] Arjun Komath
[美] Atticus White　[美] Isaac Madwed
译　林　昊
责任编辑　朱　巍
执行编辑　温　雪　赵瑞琳
责任印制　彭志环

◆ 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
三河市海波印务有限公司印刷

◆ 开本：800×1000　1/16
印张：10
字数：237千字　2017年9月第1版
印数：1－3 000册　2017年9月河北第1次印刷
著作权合同登记号　图字：01-2016-5338号

定价：45.00元

读者服务热线：(010)51095186转600　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

站在巨人的肩上

Standing on Shoulders of Giants

iTuring.cn

站在巨人的肩上

Standing on Shoulders of Giants

iTuring.cn

前　言

什么是React Native？

你可能已经听说过React，这个由Facebook开发的框架已经流行多时，如今成为了现代Web开发的标准。React使得开发者可以编写和构建声明式组件，清晰地理解应用架构。React不会给开发者的其他技术栈造成冲突，可以与任意后端技术甚至其他前端技术搭配使用。

你可能在想："哇，这听起来很不错，但用在移动端会如何呢？我是否也能用React来写移动应用？"

实际上，有两种用React开发移动端的方式。React本身可以在移动Web上运行，这就意味着所有标准React元素都可用。然而这样做本质上还是开发Web应用，所有基于Web搭建的应用所面临的性能和权限问题，React应用同样会遇到。

如果可以在移动端用React进行原生开发，那么……

幸运的是，真的可以这样做！如果你打算开发原生移动应用，用React Native吧。React Native把React的模式与范例带到了原生移动开发中。它的思想与React相似，不对现有的应用架构和技术栈造成冲突。开发者可以把React Native与几乎所有的技术进行组合，搭建出满足各种架构模式的应用。

尽管两者非常相似，React和React Native之间依然有很多明显的差别。首先，React Native自带的基础组件库与React的不同。React Native的开发过程不需要编写`div`和`span`标签，而要使用视图以及文本组件，并且有权限调用诸如地理位置、通知推送、加速器数据、设备振动等大量原生工具。相比在移动浏览器中开发React应用，React Native赋予了开发者更多可能。

这时你可能会觉得，React Native与Apache的Cordova类似。诚然，两者的思想非常相似，都共用Android和iOS两个平台的代码库。然而，Cordova运行在WebView中，通过调用API获得原生级别的功能。React Native组件会被渲染成原生部件，可以为移动应用提供真正的原生体验，而Cordova应用在遇到滚动这样高强度的UI交互场景时，可能会发生崩溃——这就是React Native的威力所在。开发者可以受益于React声明式UI的编程风格；同时，所维护的用户界面提供的极速体验，能够与任何原生移动应用相媲美。

React Native的社区活跃且增长迅速，并且已经被Facebook等许多公司用于生产环境。本书将致力于帮助你理解React Native，以便弄清React Native是否适用于自己的应用。

我应该使用React Native吗？

你是否属于小型团队，并且你的团队想要为iOS和Android两个市场开发应用？

比起为iOS和Android两个平台开发各自的应用，React Native可以让你一次性开发跨平台应用，这样更省时省力。你可以更快地发布应用，从而将更多的时间和精力集中于用户体验，无需操心平台的差异性。

你是否正在使用NodeJS和其他JavaScript前端技术？

React Native由纯粹传统的JavaScript和JSX语法编写。开发React Native能让你在不同的应用环境中切换自如，无需因为编程语言的不同而改变开发环境。你还可以从npm以及其他资源仓库获取第三方JavaScript库用于应用开发。

你是否希望，无需等待Apple、Google或者Amazon的审核过程，就可以为自己的应用发布更新？

微软的CodePush服务集成了React Native的支持。用React Native进行开发，可以直接把JavaScript更新包部署到应用上，无需等待任何第三方服务的审核。比起正式发布的流程，你可以更快地修复bug和新增特性，并提供更广泛的兼容性。

你是否希望，能够维护更小的代码库，以便更清楚地构思、更快地发布应用？

React Native共用了iOS和Android的代码库。利用React Native，再小的团队也能够做更多的事情。除此之外，Web开发者可以立刻加入移动应用的开发，进而了解移动端的原生环境。

值得使用React Native的理由远不止上述这些。要弄清React Native是否适合你，唯一的方式就是了解它能够给你带来什么。本书将会带你过一遍React Native的代码库，并展示真实的应用是如何开发出来的。

阅读前提

本书假定你至少了解基础的JavaScript知识，并且熟悉ES2016语法。不过，即使不熟悉这些，大多数人在阅读本书的过程中也能学会。

阅读本书不需要有React的开发背景。React Native和React不同但存在交集，书中将会详细描述它们相关的部分。

本书内容

本书将指导你如何开始编写React Native应用，书中通过几个企业应用的实例，让你对当今业界如何使用React Native有一些认识。

第1章简要概述移动应用开发的现状，并介绍React和React Native的诞生。第1章将介绍JSX语法以及在React Native应用运行的过程中发生了什么，然后介绍一些React Native组件以及它们的生命周期和能力，此外还有部署过程与原生模块的使用。

第2章涉及原生组件的方方面面：用JavaScript、Java和Objective-C创建自定义原生组件；从组件、常量、事件中进行异步调用；链接第三方库。

第3章介绍Myagi应用。Myagi为零售销售人员提供训练平台。你在了解Myagi的过程中会接触到Marty.js、深度链接以及环境配置。本章将带领你实现一个自定义的构建脚本，以及学习如何在iOS、Android、Web应用之间共享代码。最后介绍维护无bug移动应用至关重要的一环——测试与质量保证，同时还将提到CodePush服务。

第4章介绍基于位置的移动聊天应用TinyRobot。你将学到用Flow进行静态类型检查，接着学习Flux、Redux、MobX以及它们的异同点，还将学习依赖注入、持久化以及应用状态管理。由于React Native是无结构化的，本章会带你了解一些设计模式，以及如何从UI中分离业务逻辑，还有如何实现UI测试。

第5章介绍Fixt，它同时为普通消费者和企业客户提供手机维修贵宾服务。通过讲解一系列基于React Native的解决方案，本章将指导你利用React Native实现特定的用途，并学习Fixt提供的React Native设备参数包。你还将了解到Fixt用React Native实现的Digits（Twitter的认证系统解决方案）。本书最后会给出进一步学习React Native的建议，并告诉你遇到难题时去何处寻求帮助。

代码示例

全书包含大量的代码示例，书中的JavaScript代码示例用ES2016语法编写，必要之处还附有注释。

关于作者

Isaac Madwed是一名全栈工程师，就职于Fixt，Fixt是一家提供手机维修贵宾服务的国际化公司，总部位于美国马里兰州的巴尔的摩市。他平时热衷于使用机器学习来控制艺术创作过程，并且喜欢整洁、模块化、声明式的编程方式。想要了解更多有关Fixt的信息，请访问网站www.fixt.co。想要欣赏Isaac的艺术作品，请访问网站www.imadwed.com。

Pavlo Aksonov是一名就职于Hippware的React Native UI软件开发人员。他是一名活跃的开源

贡献者，开发了React Native路由库Flux以及许多其他组件。Pavlo关注软件架构设计，并有超过15年的Web和移动端开发经验。你可以通过Twitter（@AksonovP）与他联系。工作之余，Pavlo喜欢打乒乓球。

Alexander McLeod是销售人员在线训练平台Myagi的CTO。加入Myagi之前，他取得了墨尔本大学的计算和软件系统学位，并曾在多家创业公司任职CTO。Alex大部分时间都在用JavaScript和Python编写Web和移动应用。工作之余，他喜欢研究VR项目、创作音乐、打篮球以及滑雪。你可以通过Twitter（@amcleodio）与他联系。

Arjun Komath来自印度，他不仅拥有计算机工程本科学历，还是一名精通多门语言的程序员。过去两年内，他一直与Web和移动端技术打交道。同样，他也是一名活跃的开源贡献者。他用React Native开发了Product Hunt的开源Android客户端Feline。本书第1章由Arjun执笔，你如果有任何疑问，可以通过Twitter（@arjunz）请教他。

Atticus White就职于波士顿的Robin Powered公司，是一名React Native、Angular以及NodeJS开发人员。该公司的产品Robin致力于让预订会议室更加简便，可以将Robin看成办公室版的OpenTable。他业余时间喜欢在实验室里研究开源项目并探索JavaScript的世界。想要了解更多有关Robin Powered的信息，请访问网站www.robinpowered.com。想要了解Atticus的更多信息，请访问网站atticuswhite.com或者通过Twitter（@atticoos）与他联系。

电子书

扫描如下二维码，即可购买本书电子版。

目　录

第1章 用JavaScript开发移动应用

移动应用程序（简称移动应用）是专门为智能手机和平板电脑这样的小型移动设备开发的软件应用，它们不用于台式计算机或笔记本电脑。iOS的App Store、Android的Play Store等应用商店拥有数百万的移动应用，截至2015年6月，仅App Store上移动应用的下载量就超过了1000亿。这些惊人的数字表明应用市场一直在持续扩张和发展，这种情况也使得应用开发领域对软件开发者产生了特殊的吸引力。开发移动应用时，有多种技术可供选择。对于原生应用，最普遍的做法就是为特定的平台做原生的开发：比如Apple的iOS、Google的Android，以及Windows Phone等平台。除此以外，我们还可以开发混合应用，以及只能在移动浏览器中运行的简单Web应用。本章将对这些技术进行比较，帮助你理解为什么应该考虑React Native这个可以用JavaScript和React开发原生移动应用的框架。

开发移动应用没有开发Web应用那么简单。我们要面对好几个独立的平台（iOS、Android等）。这些平台的编程语言、原生组件和应用架构等都不一样。由于这些差异，即使是相当有经验的iOS开发者，可能也无法开发Android应用，因为Android的生态系统对于他而言完全是一个新的领域。

如果你是iOS或Android原生应用的开发者，在开发过程中很可能会遇到以下几种情况。

- **编译应用**：即使每次的改动都非常小，也需要重新打包整个应用，才能在模拟器或者真实设备上看到结果，这个过程严重拖慢了整个开发进度。
- **原生并不简单**：就算定义视图或布局这么简单的操作，也需要大量的代码。
- **一次只能给一个平台开发**：Android应用无法在iOS上运行，反之亦然。因此，为了给多个移动平台开发应用，你需要掌握不同的技术栈和工具集，而这一切只是为了写出一模一样的应用。
- **更新缓慢**：想象一下这种场景，你在生产环境中发现了一个bug，尽管经过调试并找到了解决方法，你也没办法将补丁魔法般地推送给下载过应用的用户。你不得不煞费苦心地打包应用、签名，最后把更新提交给应用商店。更新包上架到应用商店之前，所有用户会一直受bug困扰，由于应用市场政策的制约，我们对这种状况无能为力。

以上这些都表明，产品开发过程不但缓慢，而且代价昂贵。但原生并没有坏处，从性能角度来看，原生应用是最好的选择，它们的UI风格更统一，并且视觉和使用体验与整个平台融为一体。

简而言之，原生应用非常棒，但要付出一定代价。

为了解决上述问题，业界已经有了多种尝试，最流行的方式便是用HTML、CSS和JavaScript等Web技术开发移动应用。所有移动平台都有浏览器，因此可以运行任何Web应用。这种尝试很不错，但并非最佳解决方案。这些所谓的“混合”应用开发起来更简单，可以多平台运行并且不需要学习Objective-C或Java。开发者的美梦成真，但这样的应用对用户不友好。这些应用不好用，至少大部分都不好用！与原生应用相比，混合应用运行起来很慢，用户界面以及用户体验都很糟糕，而且在用户体验方面与原生应用完全没有可比性。

这时就轮到救星React Native登场了。它在两个领域都是最好的选择。在接下来的章节中，你将会看到，React Native把Web开发和原生开发完美地结合到一起。在开始编写React Native应用的壮丽旅程之前，先来了解一些构成React Native的基础概念，搞清楚Facebook如何以及为何要开发React Native。

1.1　过去

在React Native之前，有Cordova（前身为PhoneGap）和Ionic这样的跨平台应用开发框架，这些框架可以渲染JavaScript、HTML和CSS所编写的WebView。这些应用没有权限调用平台的特定组件，而是用HTML、CSS和JavaScript模拟这些组件。以Android平台的导航抽屉为例，它有很多基于Web的实现，但没有一个能达到完美的原生体验（如酷炫的汉堡包菜单翻转动画http://i.stack.imgur.com/tErny.gif），总是存在某些缺陷。用户体验以及应用的整体质感不可能像那些为宿主平台原生编写的应用一样优秀。从性能角度看，原生应用可以多线程运行，而Web平台的一切只能在主线程上运行。原生应用对手势的处理更加完善。这样的对比不胜枚举。

1.2　现状

React Native起源于2013年夏天的一次黑客马拉松项目。下面是引自Facebook的一段话。

> React Native的思想便是，我们能够把Web开发中各种深受开发者喜爱的东西带到移动开发领域，比如快速迭代、用一支团队开发整条产品线。这就意味着我们能做得更快。

在React Native诞生的早期阶段，用它开发的首批项目之一是个新闻订阅应用的原型。2014年7月，Facebook工程师想要为广告管理开发一个独立版本的iOS应用，唯一的难题在于广告团队里没有一个工程师有iOS开发经验。他们发现React Native是当时的完美选择，尽管那时它还处于原型阶段。接下来的数月，Ads Manager产品团队和React Native团队合作开发了第一个完全由React Native驱动的应用，它的体验与原生iOS应用一样棒。

这个iOS应用的成功使他们突破了正在构建的产品的极限，接着他们在伦敦创立了React Native Android团队。他们想要iOS版Ads Manager的代码在Android上运行，而且真的做到了。2015

年1月，Android版Ads Manager的可用原型开发完成，尽管这个应用从性能角度来看还不够好，少了很多特性，但它证明了未来有React Native的用武之地。

2015年1月的React.js大会，React Native的首个公开预览版亮相（https://www.youtube.com/watch?v=KVZ-P-ZI6W4）。2015年3月的F8开发者大会，Facebook把它开源提供给所有人。

2015年2月Facebook Ads Manager的iOS版发布，接着2015年6月他们又发布了Android版的Ads Manager。令人吃惊的是，这两个应用共享了约85%的源代码。

如今，React Native迅速被人们接纳，这一点业界有目共睹，它已成为最好的跨平台原生移动应用开发框架之一。2016年F8大会期间，微软和Facebook共同宣布，React Native新增了对通用Windows平台（UWP）的支持（https://blogs.windows.com/buildingapps/2016/04/13/react-native-on-the-universal-windows-platform/#O25TpvFJPKjSS67Z.99），这意味着你可以用React Native为Windows Phone、个人计算机甚至Xbox和HoloLens开发应用。

据我们所知，React Native并不是首个用Web技术开发移动应用的框架，但什么原因让它从现有的框架中脱颖而出并做得更好呢？为了理解这些问题，需要对React进行深入的探究。

1.3 React的起源

React，这个用于构建用户界面的JavaScript库，就是React Native的核心。为了理解React，先要熟悉几个概念。第一个概念，声明式编程范式（范式就是计算机程序架构与组件的构建风格），用这种范式表达计算逻辑时不需要描述控制流程。简单地说，声明式编程就是你编写代码描述想要做什么，而不是怎么做。

第二个概念，异步，大多数JavaScript开发者已经很熟悉。同步是指“按顺序执行一段代码”，代码语句一行接一行地执行。这意味着每行代码都要等待前一行执行完成。异步代码让代码语句从主程序流程中脱离，主程序代码在异步调用之后立即继续执行，而无需等待异步代码完成。

> React具有**声明式**、**异步**、**响应式**的特性，使代码**可预测**，并让我们更有**把握**进行快速迭代。

1.3.1 为什么选择React

HTML编写的Web应用中有文档对象模型DOM。DOM通过对象的形式来展现结构化文档。对于Web开发者来说，文档即HTML代码，DOM又称作HTML DOM，HTML的元素在DOM中叫节点。Web浏览器负责处理DOM的具体实现，并提供API接口以便对DOM进行遍历和修改。这样我们就能用JavaScript和CSS与DOM交互，比如查找节点并修改内容、移除节点、插入新节点。无论何时想要动态改变网页内容，只要通过API接口修改DOM即可（如今的DOM API几乎实现了跨平台和跨浏览器的兼容性）。

不过，关键问题在于DOM没有对动态创建UI进行优化。尽管可以用JavaScript和jQuery库操作DOM，但大量的操作就会引发性能问题。那么React是如何解决这个难题的呢？

React的开发者采取了虚拟DOM的做法，虚拟DOM更加轻量，对真实DOM进行了抽象化，而且独立于特定浏览器的具体实现。每当触发需要改变DOM的事件时，React会创建一个新的虚拟DOM树，并将其与已有的树进行对比，计算出最少的DOM变化集合，把它们放入队列再全部批量执行，接着重新渲染视图。这种做法没有把重负荷操作全部加在真实DOM 上，因此比直接操作DOM快了很多。React的这种行为没有采用脏数据检查（dirty checking，持续检测模型变动），而是利用观察者模型进行变动检测，通过差分算法（diffing algorithm，http://calendar.perfplanet.com/2013/diff/）判断最少的DOM操作，因此很有效率。

> React为可变命令式DOM接口编程提供了**声明式**封装。声明式的编程方法只需**描述程序应该达成什么目的**，而不用关心程序应该怎么运行。然而命令式编程需要逐步编写程序，将输入值计算成期望的输出。

- **开发简单，声明式编程**：只需告诉React，你希望应用长成什么样。按照设计稿编写声明式视图，定义应用的状态。React会根据应用状态，仅更新、渲染对应的组件，非常高效。这让代码的编写和维护变得非常容易，同时更具可预测性，更容易调试。
- **组件化开发**：Facebook宣称，“用了React，你只需要开发组件”。开发一整套组件，再把它们拼装成应用。

现在我们知道了React是什么，接下来简要了解一下这些魔法背后的原理。

1.3.2　React的工作原理

编写React应用的首要任务便是开发组件，组件属于视图部分，同时也定义了应用状态，而数据层内容取决于当前渲染的视图。

- **将UI拆解成组件**：组件驱动开发，是指将代码拆分成组件，理论上组件只负责一件事。有个很简单的技巧，称为单一职责原则（the single responsibility principle），指一个组件理论上只应该做一件事。如果组件需要进一步扩展，就应该将它拆解成更小的子组件。这样可以让代码更容易理解、维护和测试。
- **交互式UI**：要使UI具有交互性，你需要触发UI背后的数据模型的改变。React中通过状态很容易做到这点。每个组件拥有内部状态、逻辑、事件处理器（如点击按钮和改变表单输入），也可以包含行内样式。一旦状态发生了改变，React就重新渲染视图。

需要注意的是，React有两种类型的数据“模型”：属性（props，英文properties的简写）和状态（state）。弄清两者的区别很重要。简而言之，如果组件有时候需要修改自身的某个特性，那么这个特性应该归类于组件状态的一部分，除此以外便是组件的属性。属性可以比作组件的静态数据，而状态是动态的，不过两者都可以触发重新渲染。

现在我们知道了React的工作原理，接着继续了解一下同样的概念如何打造出React Native。

1.4 为什么选择 React Native

React最神奇的地方在于，它被设计成能对任意的命令式视图系统进行封装，不仅限于DOM。因此Facebook的工程师某天就想到，为什么不用React来封装真正的原生移动UI呢？React Native就这么诞生了！

正如React用虚拟DOM产生的魔法一样，React Native通过原生宿主平台的API也实现了一样的效果。React Native应用借助宿主平台上Objective-C语言（iOS平台）或Java语言（Android平台）的UI库，渲染真正的原生UI组件，不仅限于WebView，这就解释了为何React Native能给应用带来更强的性能、更贴近原生的视觉感受以及使用体验。

> React Native允许开发者通过JavaScript函数的代理，直接调用原生模块。

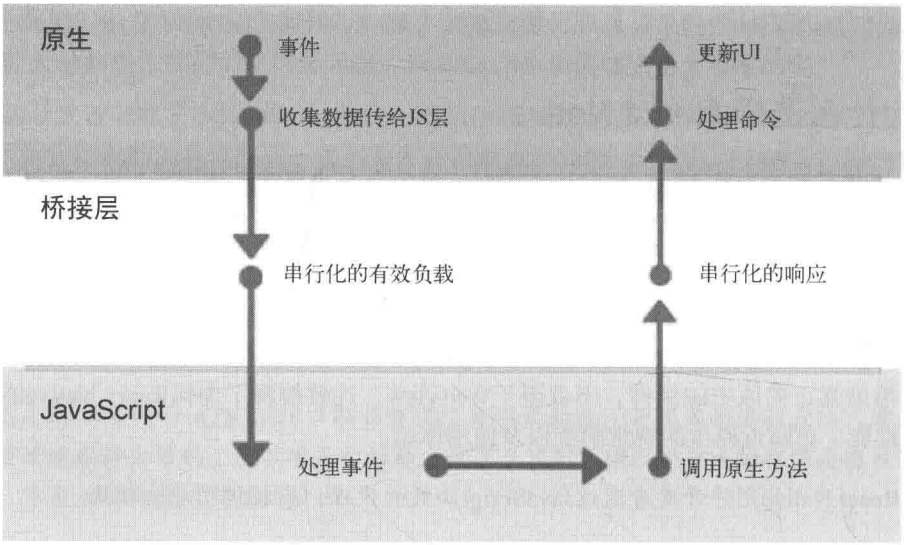
从性能角度上分析，React Native把所有应用代码和业务逻辑从主线程转移到后台线程运行。它可以批量处理要原生执行的请求，等控制权转让给主线程时再异步执行。React Native会分析你的UI，将最少的数据传给主线程（又称UI线程）以便用原生组件进行渲染。

> 使用React Native，你会得到原生的用户体验以及Web的开发体验。

- **简单易学**：如果你曾经开发过移动端，你可能会惊讶于React Native如此简单易用。
- **快速迭代**：不用等待应用构建，只要按下刷新快捷键，所有改动会立刻在应用中呈现。
- **智能调试**：与React一样，React Native也会在报错时抛出简明扼要的描述信息。
- **原生模块**：React Native的设计目的就在于，让你能够编写真正的原生代码，并在自定义原生模块的帮助下获得平台提供的完整能力。
- **一次性学习，全平台开发**：同一支工程师团队可以为任何平台开发应用，不需要为每个平台学习不同的基础技术。

1.5 React Native 的工作原理

原生代码与JavaScript代码通过桥接层进行交互，这是一个异步的批量串行处理过程。桥接层介于原生层和JavaScript代码之间，正如它的名称一样，它的作用很像桥（bridge）。用户输入、计时器、网络请求和响应等事件注册在原生代码中。React Native在原生层收集事件产生的数据，串行处理后通过桥接层传给JavaScript层。JavaScript层拿到数据后处理并生成一系列指令。这些指令由整型、字符串等数据类型构成，同样经过批量串行处理后传回原生层。桥接层的原生端决定哪个原生模块负责处理传回的指令并调用相应的方法，同时在需要的情况下更新UI。这种架构提升了React Native应用的性能，并且让React Native应用能以每秒60帧的速率运行。



React Native需要通过JavaScript执行环境运行JavaScript代码。在iOS和Android系统的模拟器以及设备上，React Native使用Safari的JavaScript引擎JavaScriptCore（http://trac.webkit.org/wiki/JavaScriptCore）。通过Chrome调试React Native时，JavaScript代码会在Chrome V8引擎内运行，并通过WebSocket与原生代码进行交互。React Native Windows应用的JavaScript运行环境是Chakra（微软Edge浏览器的JavaScript引擎https://github.com/Microsoft/ChakraCore），UWP（通用Windows平台）应用包不需要添加任何额外的二进制文件就可以使用Chakra引擎。

运行 React Native 应用时发生了什么

下面来看看React Native应用启动时发生了什么。启动应用时有以下三个任务并行完成。

- 加载JavaScript打包文件，React Native的打包工具会像Webpack和Browserify一样把代码连同全部依赖打包成单个文件。
- 与此同时，React Native开始加载原生模块。一旦某个原生模块完成加载就在桥接层注册，桥接层确认该模块。此时整个应用便知道该模块已可用并能创建该模块的实例。
- 启动JavaScript虚拟机，提供JavaScript代码的执行环境。

一旦原生模块和JavaScript执行环境准备就绪，应用就会加载JSON配置文件。配置文件中包含了模块数组、常量导出模块以及模块的方法。这个文件的重要之处在于，当你请求依赖的原生模块并调用方法时，JavaScript会读取它并在执行环境中创建对象。

下一步，执行JavaScript打包文件。创建Shadow视图（负责计算布局的独立线程称为shadow队列）来渲染应用的布局。Shadow队列把flexbox这样的属性转换成绝对位置和大小。与此同时，创建原生视图来渲染应用。最后结合两者将整个应用渲染到屏幕上。