

# 写给移动开发者的 React Native 指南



wingjay (/u/da333fd63fe5) [+ 关注](#)

2017.03.13 20:23\* 字数 6579 阅读 41519 评论 42 喜欢 282 赞赏 1

(/u/da333fd63fe5)



# React Native

## 前言

React Native 诞生于 2015 年，名副其实的富二代，主要使命是为父出征，与 Apple 和 Google 抗衡，为开发者带去一套跨平台、动态更新的 Javascript 框架，口号是：Learn once, write anywhere：Build mobile apps with React。在试图推翻 Android 和 iOS 压制的同时，还提携了一把自家兄弟：React。

从诞生之日 React Native 就充满了期待和争议。期待是无数开发者希望不用忍受频繁发版的噩梦，也不用同时为两个平台开发业务逻辑几无差别的两个 App；争议是 React Native 真的能以一己之力救大众于水火吗？React Native 在跨平台时还能保持良好的用户体验吗？

当然我们知道，这种问题向来都是仁者见仁，智者见智。比起一味的疑惑、争论，还不如来好好看看这货究竟是个啥？甚至自己动手来玩一把。

本文主要针对两类读者：

- 想要入门 RN 的人，在阅读官方文档前先对 RN 形成一个整体的印象
- 对 RN 心存好奇，在犹豫是否要入坑的开发者，可以通过本文对 RN 更客观全面的认识

## 目录

- React Native 好在哪
  - 跨平台 + 动态更新
  - 代码复用
  - RN vs Weex
  - RN vs Hybrid
  - RN 劣势



- React Native 运行机制
- RN 开发环境搭建
- 引入 React Native
  - Build from Scratch
  - 集成到已有项目
- Javascript、React 及 ES6、JSX 语法
- UI 层
- 网络请求层
- Debugging 调试
  - In-App 报错
  - Console.log
  - 大杀器：Chrome 逐行调试
- 从 JS 调用 Native 方法或显示自定义 Native View
  - Native Modules：JS 里直接调用 Native(Java/Swift) 方法
  - Native UI Component：JS 里直接调用自定义的 Native View
- React Native 适合你吗？
- 为什么要写这篇文章

## 一、React Native 好在哪

下面我们来看下 Hybrid 及 React Native 等开发模式包含了哪些常规移动开发所不具备的优势。

### 1. 跨平台 + 动态更新

传统的客户端开发模式是怎样的呢？

Android 与 iOS Team 分别编写客户端代码，打包，分发到 Play Store 和 Apple Store，通过更新 JSON 数据来更新页面。

不过，当客户端发生严重问题而服务器上无法 quick fix 时，就不得不重新发版。

对国外 Android 市场而言还好，因为能通过 Play Store 快速更新；国内 Android 市场则由于分发渠道太杂，很难及时把新版本立即推送给所有用户，当然这也是为何热修复技术在国内盛行而国外冷清的原因；而 Apple Store 则需要一定的审核时间，而且最近又在抓 iOS 热修复框架如 JsPatch、Rollout 等。

相比而言，Hybrid 和 RN/Weex 模式除了能下发 Json 数据来刷新界面内容，更能直接下发业务逻辑代码，直接实现整体 App 的更新。而且，它们不用在乎 Android 和 iOS 两个平台，因为一份 JS 代码写好后，把 JS Bundle 放在服务器上，所有的客户端立即更新。

### 2. 代码复用

一般而言，同一款产品的 Android 和 iOS 两端除 UI 有些许不同外，多数业务逻辑几乎完全一致，这便造成了人力的浪费。

而最近 Instagram 的官博 React Native at Instagram

([https://engineering.instagram.com/react-native-at-instagram-](https://engineering.instagram.com/react-native-at-instagram-dd828a9a90c7#.ugk9ncjyz)

dd828a9a90c7#.ugk9ncjyz) 一文中已经提到，利用 RN (React Native 缩写，下同) 开发



的 feature 可以实现 85% - 99% 的代码复用率。这意味着我们可以用更少的人力成本来达到相同的效果。

3. RN vs Weex

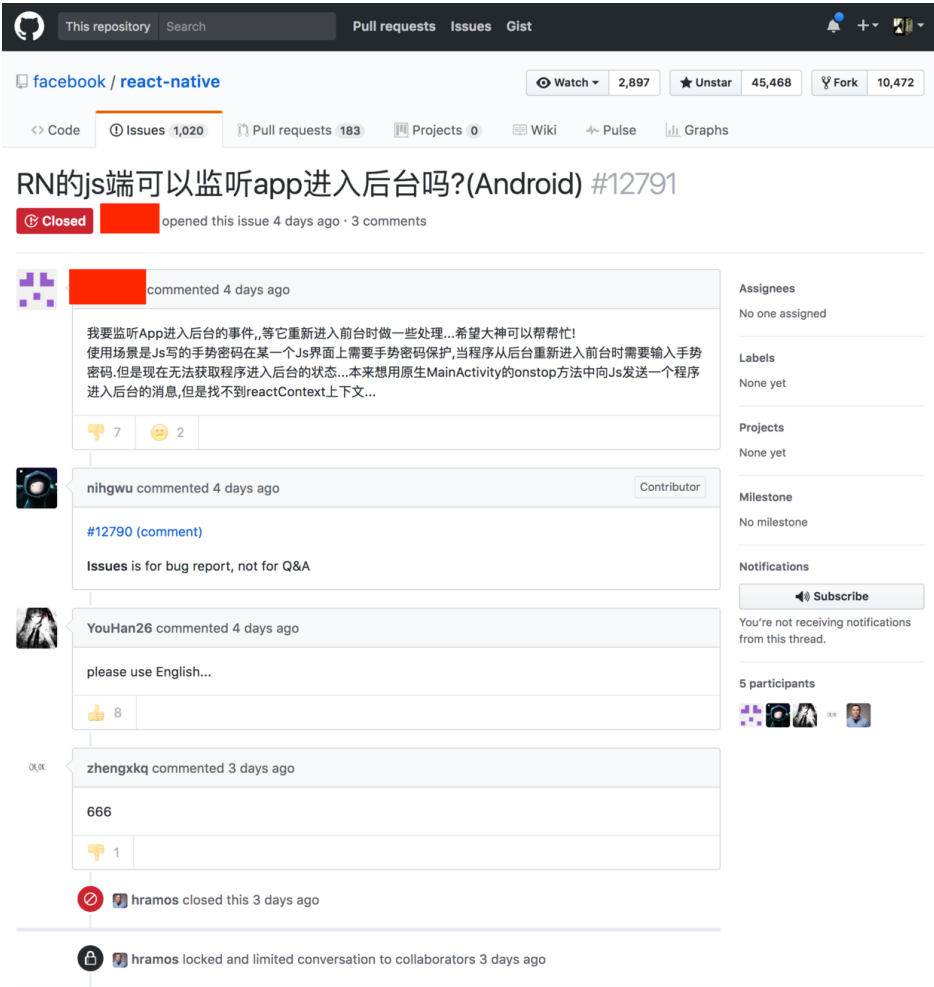
实现上面的效果有两种开发框架：混合开发框架 Cordova 和基于 Javascript 的 React-Native、Weex 框架。

下面我从自己的实践经验出发做些比较，也欢迎读者提出自己看法。

最开始觉得 RN 的学习成本比较大，所以首先考虑了 Weex 框架 (<https://github.com/alibaba/weex>)，据说是阿里巴巴良心出品。不过在尝试后不得不选择了放弃，原因有这几点：

- Bug 较多。我们最先测试了 最基本 的 ListView，在 iOS 运行良好，而同样的 Demo 代码到了 Android 这边的 下拉刷新 就出现了问题，这使得我们开始警惕；
- 社区、文档弱，GitHub Issue 基本是中文。当然我毫无歧视中文之意。我认为，一套项目开源是真正意义是希望借助开源社区的力量，一起来完善改进，因此要优先推崇英文，使项目国际化，得到全世界开发者的共同支持，这样才是可持续的模式。而 Weex 的 Issue (<https://github.com/alibaba/weex/issues>) 里放眼望去基本 90% 都是中文，无论提问者还是项目维护者。这一点直接把国外优秀的开发者拒之门外，也很难让我看到多么长远的未来。

下面是摘取的 RN 里的一则中文 issue：



Issue is for bug report, not for Q&A

- Contributor 差别。因为上面一点，Weex 的 Contributor (<https://github.com/apache/incubator-weex/graphs/contributors>) 只有 91 个人，而 React-Native 的 Contributor (<https://github.com/facebook/react->



native/graphs/contributors) 有 1214 人。Contributor 是用来干嘛的？除了支持新功能，还有就是修复 bug 啊。Weex/RN 都是希望一统 Android + iOS 的，这么伟大的目标，这么艰巨的工程，不是几个人可以轻轻松松搞定的。

- 公司背景（来自YY）。大家都知道 RN 来自 Facebook，Weex 来自阿里巴巴。如果想一窥它们的未来，需要先想一下这种技术对他们各自的意义。大家都清楚，Facebook、Google、Apple 是当今当之无愧的巨头，在移动互联网这波浪潮里，Google 掌握了 Android 法器，Apple 控制了 iOS 神器，Facebook 呢？并没有这些系统级入口。当然 Windows 的经历也让 Facebook 并不那么倾向去开发一个新的移动操作系统来竞争。那怎么办？React Native 应运而生，打出的口号就是：**Learn once, write anywhere**。什么意思，没错，就是明确告诉你学一次就可以同时开发两个平台了。这一点可一直都是移动端开发人员和创业公司的理想。有人说了，Apple 这么强势，RN 要是太嚣张，分分钟把你禁掉。这时我们就要来看看 RN 的 Showcase (<http://facebook.github.io/react-native/showcase.html>) 了，哪些 App 应用了 RN 呢？Facebook, Instagram, Airbnb, Walmart, QQ, 京东等，这回 Apple 要禁 RN 就要稍微掂量下这些大厂的意见吧。

当然，我是很希望国内也能推出优质的开源项目来和国外大厂抗衡的，不过真正优质的大型开源项目往往除了开发者的个人能力，和公司的战略和制度关系也很大。

#### 4. RN vs Hybrid

这里的 Hybrid 开发主要针对 Cordova 框架，其实在放弃 Weex 之后我们还是没考虑 RN，而是转过去了解 Cordova，不过做了大致了解后也放弃了。主要硬伤有两点：

- 性能短板。大家知道 Hybrid 是基于 WebView 的，在 Android 上的性能缺陷非常明显；而 RN 是利用 JSCore 转化成 Native 运行的，性能相对而言好不少；
- 用户体验。了解移动产品的人都知道用户体验的重要性，RN 的体验和原生的几乎没有差别，而 Webview 的实现是网页开发思路，体验会相差很大。

性能和用户体验是移动 App 的命根子。

因此，综合考虑下来，我们还是决定相信 Facebook 并采用 RN。

#### 5. RN 劣势

上面我提到了 RN 的一些优势，不过作为开发者更加需要明确其劣势，我总结了大概有以下几点劣势：

- 学习成本。Weex 的写法就是类似常规的 Html/JS，对于前端人员来说很容易上手，就算非前端人员来说也花不了多久。而 RN 是在 React.js 上进行改进形成的一套语法，和常规前端差别较大，因此需要好几天的学习适应。当然我觉得优秀的程序员的基本素质之一就是能快速学习、练习并熟练一种新语言的。我个人的话大概花了两三天的时间已经能完成一套涵盖网络、JS与Native通信的页面了，对于 React.js 语法也上手很快。
- 安装包 Size。对于 iOS 而言影响不算很大，对于 Android 来说，我尝试后发现引入 RN 会给 apk 带来 6MB 左右的增幅，不过利用 `split apk` 的技术就能缩小到 1MB 左右的增幅。
- 首次加载耗时。大家知道 RN 需要从服务器下载 JS bundle，然后在本地转化成 Native code 运行的，所以在第一次打开 App 时需要花费一些时间进行下载和刷新。当然我们可以在发布 client 时内置一个写好的 js 文件在本地作缓存用。

## 二、React Native 运行机制



对于一个用 RN 搭建的移动 App，在启动后会从服务器下载最新的 JS Bundle 文件，然后由本地 JavascriptCore 引擎对 JS 文件进行解析，并利用 Bridge 映射到对应的 Native 方法和 UI 控件。得到的效果是：

1. 同样的 RN 代码，下发到 Android 和 iOS 不同平台中，会自动调用对应 Native 的 UI 控件，保证了各平台用户体验的连贯性；
2. 开发者就算是移动端小白，只要有 Web 基础，通过编写一套 RN 端代码就可以同时完成 Android 与 iOS App 的开发；
3. 由于可以利用 JS bundle 同时下发数据和业务逻辑，这意味着你可以像 Web 开发一样，实时迭代更新你的移动端 App，无需在了解各自平台的热修复技术；
4. Native Modules，这是 RN 强大的一个扩展性，允许你通过简单的代码就能实现在 JS 里直接调用你自己的 Native 方法；
5. Native Components，如果你自己实现了一些复杂的 Native UI 组件，而这些组件尚未被 RN 支持，你可以利用 Native Components 快速把原生组件引入到 RN 中并可以直接在 JS 里更新这些组件的状态。

### 三、RN 开发环境搭建

首先 IDE 方面，RN 推荐了一些工具：

- Nuclide (<https://nuclide.io/>) 是 Facebook 内部用来开发 RN 的工具，Debug 功能强大。只不过这是一款 Atom (<https://github.com/atom/atom>) 的插件，意味着你必须先安装 Atom，再来安装这款开发插件；
- Deco (<https://www.decosoftware.com/>) 是专为开发 RN 诞生的工具，可以快速搜索开源的第三方 RN 组件并直接插入到代码中，用 MacOS 的同学可以尝试下。我本人最开始也是试用这个，上手简单、小巧简洁。不足的是功能有点简单，无论是 Debug 功能还是代码检查之类的都不具备；
- Sublime (<https://www.sublimetext.com/>) 可以通过第三方包来达到不错的开发效率，各方面还算可圈可点；
- Visual Studio (<https://code.visualstudio.com/>) 这款也是蛮强大的 IDE，之前有用过的小伙伴可以试一下。

本人的话目前采用的是 Sublime，因为个人常用 Sublime，而且第三方插件很丰富，轻量方便。下面简单说下配置，感兴趣的小伙伴可以看下。

1. Babel (<https://packagecontrol.io/packages/Babel>) 用来高亮 React JSX 语法，支持 ES6，而 React-Native 就是搭建在 React.js (<https://facebook.github.io/react/>) 基础上的；
2. React-Native-Snippets (<https://github.com/Shrugs/react-native-snippets>) 可以快速生成 RN 的一些模版代码；
3. ESLint (<http://eslint.cn/docs/user-guide/configuring>) 超级强大的 Lint 工具，支持 ES6、JSX 语法检查，而且还有 React 和 RN 的插件，比纯粹的 JSXHint/JSLint 都强大；

当然，用 Atom 的小伙伴自然要首先考虑 Nuclide (<https://nuclide.io/>)。

### 四、引入 React Native

引入 RN 有两种方法：从零构建；集成到已有项目。

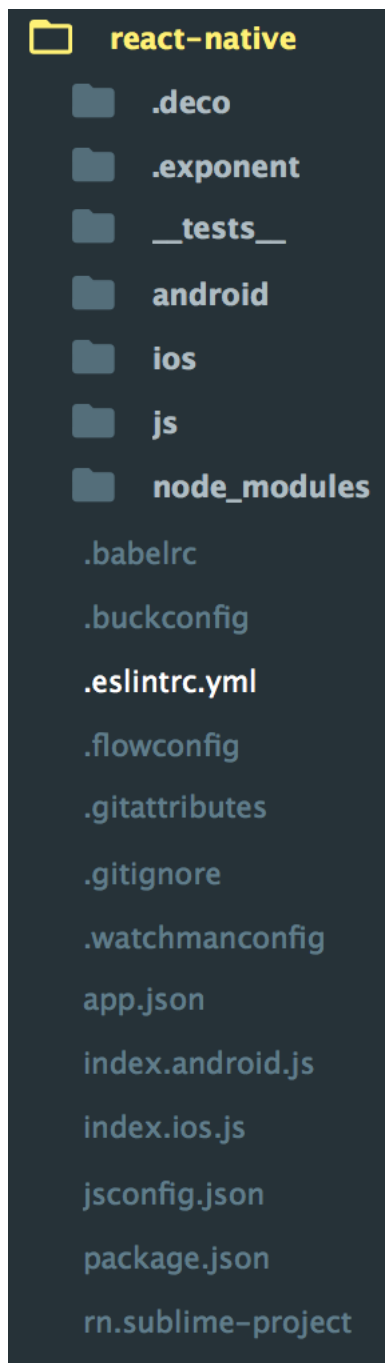
#### 1. Build from Scratch



先说第一种，从零开始构建，比较简单，遵循官方文档 [Getting Started](http://facebook.github.io/react-native/releases/0.42/docs/getting-started.html) (<http://facebook.github.io/react-native/releases/0.42/docs/getting-started.html>) 基于你自己的操作系统和平台一步步安装相关的依赖，然后利用如下命令：

```
react-native init AwesomeProject
```

你就创建好一个 RN 工程项目了，结构如下：



RN 目录结构

里面有四个文件夹：

- `android / ios`：各自存放了一个相关平台的工程 project，可以直接下拉 JS Bundle 并运行，对于移动端小白而言可以不用管里面的具体实现；
- `node_modules`：里面是自动生成的 node 依赖之类的文件，通过读取 `package.json` 里的配置来生成；
- `js`：这个文件夹最为重要，我们的开发都在这个文件夹里，把写好的 `js` 文件打包下发给 client 就会自动生效。



## 2. 集成到已有项目

有很多公司是希望在现有 App 的基础上集成 RN 来开发一些特定的 Feature，这种情况就不能参考上面的方法了。在 RN 的官方文档里有一节 [Integration with Existing Apps](http://facebook.github.io/react-native/releases/0.42/docs/integration-with-existing-apps.html) (<http://facebook.github.io/react-native/releases/0.42/docs/integration-with-existing-apps.html>)，只需要按照一步步做即可。

以 Android 为例，大概要做以下几步：

1. 添加 gradle 依赖：`compile "com.facebook.react:react-native:+" // From node_modules.;`
2. 创建空的 Activity，指定 JS bundle 和入口 Component 名字即会自动在这个 Activity 里去加载 JS bundle 文件；
3. 在 Activity 里监听 `onBackPressed` 事件，用来与 JS 端协作处理返回键点击事件。
4. 启动 server，运行 App 即可。

总之需要说明的是，即使是移动端小白，也可以遵循文档里的指示完成这一步。接下来的大部分时间只要关心 JS 端开发就行了。

## 五、Javascript、React 及 ES6、JSX 语法

我们知道 RN 采用了 React 和 ES6 的语法，所以我们必须先对这些语法有一定了解才能去读 RN 的代码。

关于 Javascript，我推荐 W3School 里的 JS 语法 (<http://www.w3school.com.cn/js/>) 和 MDN 里的 JS 手册 (<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript>)，大家只要对一些基础语法做些了解就可以。

关于 React，我推荐 阮一峰 (<http://www.ruanyifeng.com/>) 写的 React 入门实例教程 (<http://www.ruanyifeng.com/blog/2015/03/react.html>)，基本上把文章读一遍，再自己动手写一遍，就能领会到 React 的大致用法了。

关于 ES6、ES7、JSX 等，感兴趣的可以看一下 RN 文档中 Javascript Environment (<http://facebook.github.io/react-native/releases/0.42/docs/javascript-environment.html>) 里提到的支持的方法，需要时再来查询也可以。也可以看 Babel 出的 Learn ES2015 手册 (<https://babeljs.io/learn-es2015/>)。

这里有一个很不错的 GitHub 项目，帮助你通过交互性的例子来快速上手语法知识：React Native Express (<http://www.reactnativeexpress.com/>)。

## 六、UI 层

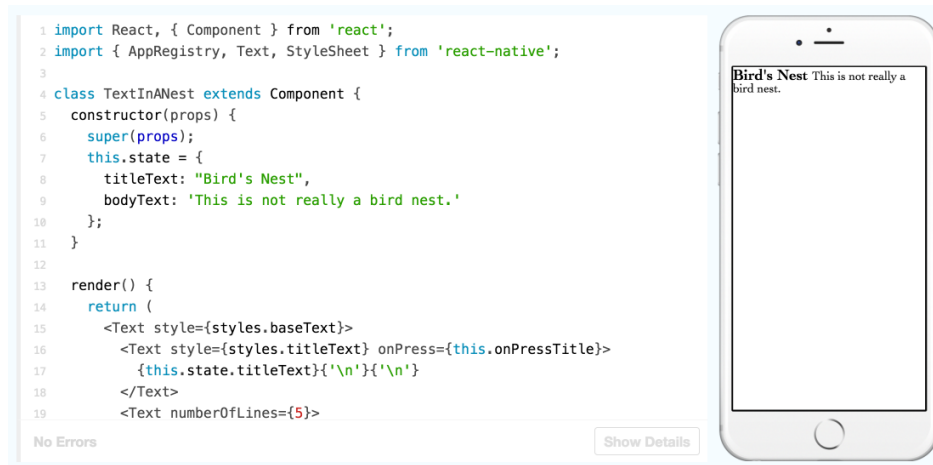
简单熟悉了 React 语法后，基本能正常阅读 RN 的示例代码了。

正式开发 App 的第一步当然就是写 UI 界面了，由于 RN 已经封装好了一套 JS 的 UI 组件，这些组件会自动在 Android/iOS 端调用对应的原生 UI 组件，因此我们只需要熟悉这些 UI 组件的用法及属性、回调方法即可。

我们可以在文档的 Components (<http://facebook.github.io/react-native/releases/0.42/docs/getting-started.html>) 看到不少组件，比如 View, Text, Button, Image, Switch，还有我们用的最多的 ScrollView 和 ListView。

在读文档时，我们可以先通过一边写代码一边读文档的方式进行，RN 非常贴心，直接在 Web 里嵌入了模拟器，我们只要修改编辑框里的代码，立即就能在右边的模拟器看到效果。这极大的降低了我们的学习成本。





Text Component

另外，在学习一个组件时，我们要区分哪个属性是某个平台特有的。比如下面两个 `Text` 的属性：`textBreakStrategy` 只会在 Android 上生效，而 `adjustsFontSizeToFit` 只可以用在 iOS 上。

**android** `textBreakStrategy?: enum('simple', 'highQuality', 'balanced')`

Set text break strategy on Android API Level 23+, possible values are `simple`, `highQuality`, `balanced`. The default value is `highQuality`.

**ios** `adjustsFontSizeToFit?: bool`

Specifies whether font should be scaled down automatically to fit given style constraints.

Platform Specific Properties

然后，如果你希望在 Android 和 iOS 里显示不同的内容怎么办呢？RN 里有一节是 Platform Specific Code (<http://facebook.github.io/react-native/releases/0.42/docs/platform-specific-code.html>)，可以有如下几种形式来进行区分：

```
if (Platform.OS === 'ios') {  
  // stuff for ios  
} else {  
  // stuff for android  
}
```

除此之外，UI 组件的用法学习就很类似常规的 HTML 标签了，只要知道其使用方式即可，甚至需要用的时候再来查文档也行。

## 七、网络请求层

学完上面的我们已经能够写出 UI 界面了，而且这套界面已经能够在不同平台上转化成各自平台的 Native UI 了。然后，我们就需要去网络层请求真实数据了。

RN 里提供了 Fetch API ([https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)) 来进行实现。举个例子，你希望通过 GET 方法去请求数据并转化成 JSON，可以通过如下代码实现：





```
fetch('https://facebook.github.io/react-native/movies.json')
  .then((response) => response.json())
  .then((responseJson) => {
    return responseJson.movies;
  })
  .catch((error) => {
    console.error(error);
  });
```

熟悉 Reactive 编程的伙伴应该对这样的语法不陌生，比如 Android 上的 RxJava (<https://github.com/ReactiveX/RxJava>)；iOS 上的 RxSwift (<https://github.com/ReactiveX/RxSwift>)；Web 上的 RxJS (<https://github.com/Reactive-Extensions/RxJS>)。上面 function 的功能就是：请求网址 <https://facebook.github.io/react-native/movies.json>，把返回的 Response 转化成 JSON object，取出 JSON object 里的 movies 字段。同时，如果发生 error 会被 catch 住。

当然，上面是最基本的 GET 请求，Fetch API 还支持自定义 Headers，更换 Method，添加 Body 等。

```
fetch('https://mywebsite.com/endpoint/', {
  method: 'POST',
  headers: {
    'Accept': 'application/json',
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    firstParam: 'yourValue',
    secondParam: 'yourOtherValue',
  })
})
```

上面构建了一个基本的 POST 请求，添加了自己的 Headers: Accept 和 Content-Type，添加了 Body。

因此看下来，RN 里的网络请求不仅具备了 Reactive 编程的简洁，也能自定义常规的 Http 请求，写法简单。

除了 Fetch API 之外，RN 还内置了 XMLHttpRequest API(俗称 AJAX) (<https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest>)，而且支持 TCP 全双工通信方式 WebSocket (<https://developer.mozilla.org/en-US/docs/Web/API/WebSocket>)。

## 八、Debugging 调试

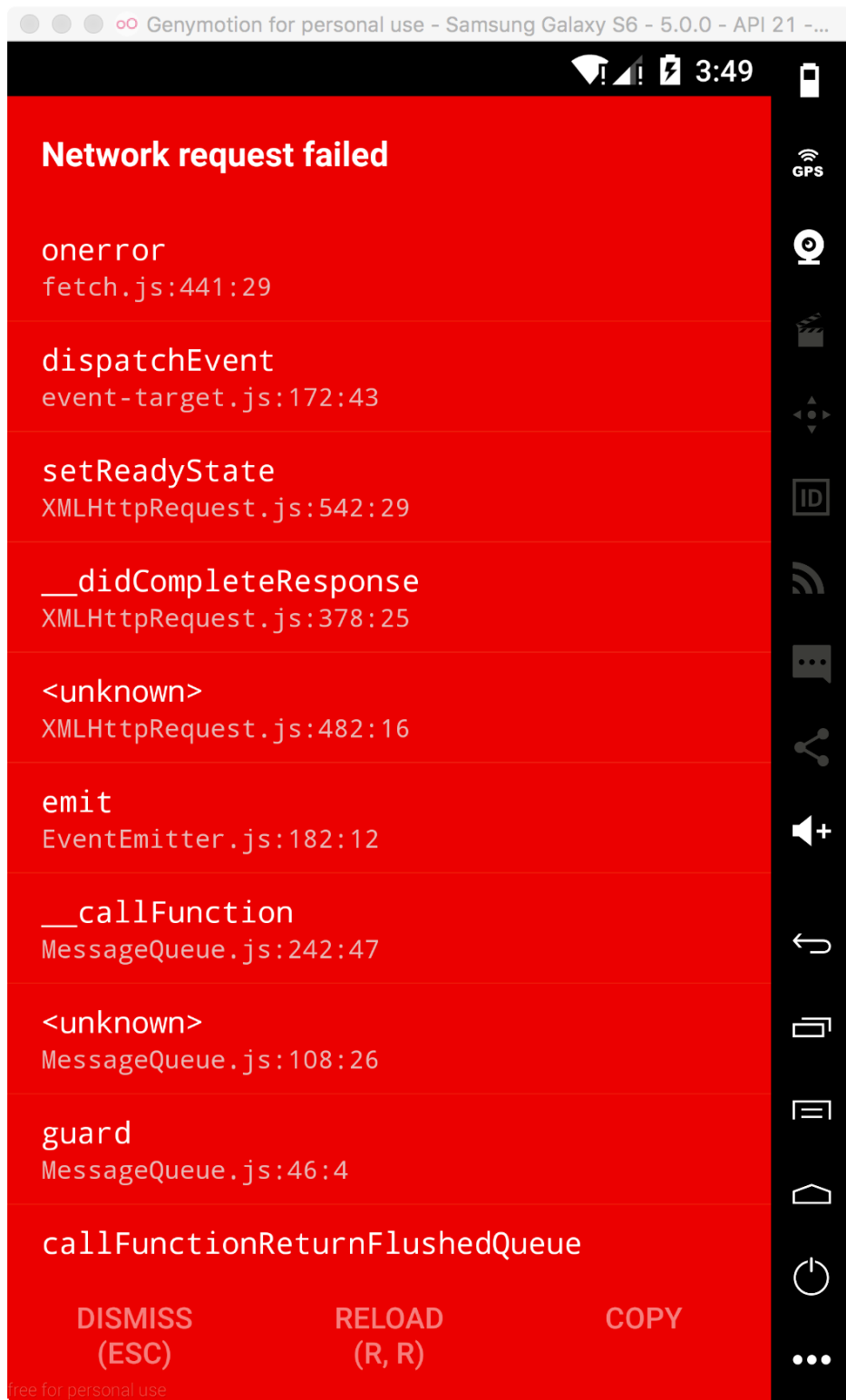
调试是很多程序员非常关注的一个环节，因为 RN 是用 JS 写完后到 Native 解释成 Native 方法来执行的，因此如果能快速调试 JS 代码是非常重要的一环。

最开始 RN 的调试功能比较弱，不过现在的 Debugging 功能在我看来还是很不错的。一般来讲可以有以下几个调试方式：

### 1. In-App 报错

RN 里默认集成了 In-App 的错误提示方式，即在 App 运行过程中会弹出全屏的报错信息呈现给你，而你也可以通过阅读具体的错误信息快速找到错误原因。通过点击这个错误信息里的某一行，会立即自动打开对应的代码。





## 2. Console.log

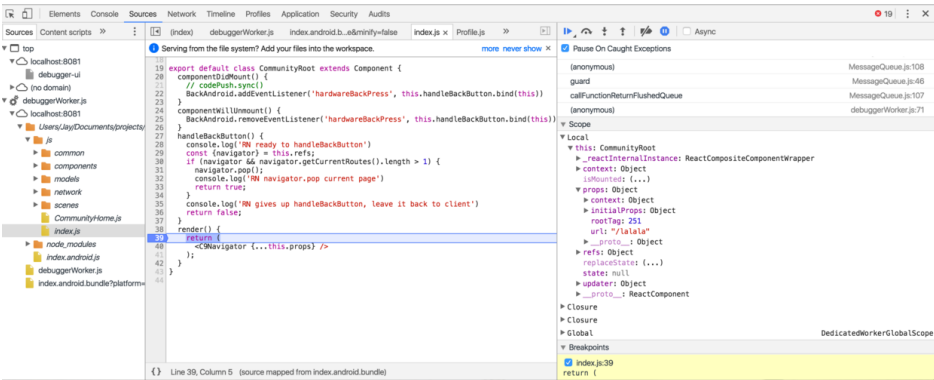
在开发 Client 时，我们一般都会用 `Log.log()` 来打印一些运行时变量的值，然后实时查看打印出来的 log 来调试，在 RN 也一样，你只要在 JS 里写一句 `console.log('this is log data')`，就会自动在 Client 的常规 log 里能看到，比如 Android 的 `adb logcat` 里就会自动打印出 'this is log data' 一行。

## 3. 大杀器：Chrome 逐行调试

这个杀器的最牛逼之处就是可以像 Client 一样，逐行调试代码！



我们来看下面一张图。从左往右。先是文件目录，我们选中了 `index.js` 文件夹，然后第二个 Tab，是 `index.js` 的内容。这里关键的是我可以直接选中某一行代码设断点。当 Client 运行到这一行时，就会在第三个 Tab 里打印出运行时环境及变量。我们可以看到 `props` 里就有我们传进去的变量值。



Chrome Debug

有了以上几种调试方式，我们几乎可以和常规的 Native 开发一样来调试 RN 代码了，不得不说 RN Team 确实牛 x 啊！

### 九、从 JS 调用 Native 方法或显示自定义 Native View

这又是另一个牛 x 之处啊。

很多人觉得 RN 限制太多，只能支持有限的 View 组件和有限的方法，难以发挥 Client 的最大性能。简单点说，在 Client 可以绘制复杂的 View，可以调用高性能 C++ 等底层代码，但 RN 却做不到。

于是，RN 里提出了 Native Modules 和 Native UI Component 两种技术。

#### Native Modules：JS 里直接调用 Native(Java/Swift) 方法

所谓 Native Modules，就是自己在 Client 写好了某些方法，由于某些原因这些方法不太方便或者无法搬到 RN 里面，那么，我们可以在 Client 把这些方法暴露出来给 RN，然后在 JS 里可以像 import 普通的 module 一样把这些 Native Modules 引入进去，直接调用。

具体的实现方法可以参考文档 iOS Native Modules (<http://facebook.github.io/react-native/releases/0.42/docs/native-modules-ios.html>) 和 Android Native Modules (<http://facebook.github.io/react-native/releases/0.42/docs/native-modules-android.html>)。

#### Native UI Component：JS 里直接调用自定义的 Native View

很多时候我们在写 Client 时，为了实现 Designer 天马行空的设计，常常需要自定义 View，即自己绘制某些系统并不提供的特定 UI。可想而知，这些 View 肯定不会出现在 RN 的 UI Component 里。

那么，我们就需要首先在 Native 层自己写好一个自定义 View，然后利用 Native UI Component 技术把这个 View 及其中某些 public 方法暴露给 RN，那么 RN 就能直接 import 进来并显示了。

具体的实现方法可以参考文档 iOS Native UI Component (<http://facebook.github.io/react-native/releases/0.42/docs/native-components-ios.html>) 和 Android Native UI Component (<http://facebook.github.io/react-native/releases/0.42/docs/native-components-android.html>)。



如果读过文档不是很理解的小伙伴可以留言，我再 post 一些 demo 代码上来

## 十、React Native 适合你吗？

这里借鉴下前段时间旧金山的 React Native 会议上的一些优劣总结给读者以参考。当然不一定对，仅供参考。

RN 的优点：

- 跨平台
- 原生的用户体验
- 开发者体验好
- 动态更新代码逻辑
- 社区强大
- 有个好爹

RN 的缺点：

- 不够成熟
- 不够稳定
- 生态系统在搭建中
- 优质的 App 需要时间打磨
- 偶尔需要写 Native 代码(也就是 JS + Swift + Java)

适合下面这些人/公司：

- 你对 JS/React 有一定了解
- Web 开发人员比 Mobile 开发人员多
- 有意愿投资精力给 RN
- App 设计不是特别区分 Android 和 iOS
- 希望热更新

下面这些人要稍微考虑下：

- 完全不了解 JS/React
- 已经有现成的 Android/iOS team
- App 设计严格遵守 Android、iOS各自设计规范
- 不想要投入时间 / 金钱给 RN

## 十一、为什么要写这篇文章

几个月前我对 React Native 也非常不看好，当然现在也没有说非常看好。或者说，写这篇文章毫无为 React Native 布道之意。

接触 React Native 主要是因为业务需要，PM 希望能够随时改动某块变化较大的模块，常规的开发提交流程往往需要较长的时间，而热修复技术本身并未得到 Google 和 Apple 的官方认可，也就是随时可能因破坏生态安全之名被取缔。

因此才考虑去了解 Hybrid 开发和 JS Native 开发模式，在了解过程中，又由于性能差、用户体验不好而放弃 Hybrid，由于社区不完善、Bug 较多等原因放弃 Weex，最终才选择了 React Native，开始学习 React、JSX等语法。

目前使用下来对 React Native 的一些个人感受：



1. 学习门槛并没有开始想象那么高。大概只花了两三天时间就熟悉了 Javascript、React 框架、JSX语法，然后就开始着手业务开发。
2. 对 Android App 的影响。React Native 会给 Android 端带来 6MB 左右的 size 增幅，不过在采用了 split apk 后就只有 1MB 左右增幅。
3. Debug 功能比较完善，至少不用担心发生问题后不知从哪下手。
4. 性能还行。最初担心的是 React Native 性能不好，但自己上手后，并没有明显感觉到很明显的 React Native 对 App 性能的负面影响，无论是 iOS 还是 Android，当然，这一点还在继续考察中。
5. 动态部署真的很不错。以前每次写好代码都要花不少时间来编译运行，而现在只要写一份代码，就可以同时在 Android 和 iOS 实时更新了，这无疑节省了生命。
6. 有待完善。当然，React Native 中确实还存在着不少问题，生态系统也还不够完善。不过我相信，这只是时间问题。

关于React Native一直以来都有很多争议。

不过我想说的是，**React Native** 所代表的跨平台、动态更新技术已经引起了全世界开发者关注，而且这种技术势必会是未来的需求和潮流。**React Native** 不一定会成功，但至少目前 **React Native** 已经是这一领域的领跑者。

而写这篇文章的目的，就是希望告诉更多开发者，**React Native** 并不完美，但值得一试。

谢谢。

wingjay



React Native学习研究 (/nb/10615743)

举报文章 © 著作权归作者所有



wingjay (/u/da333fd63fe5)

写了 72752 字，被 2950 人关注，获得了 4337 个喜欢  
(/u/da333fd63fe5)

+ 关注

阿里各部门岗位内推欢迎联系我。Big Fan of Full-Stack. GitHub: <https://github.com/wingjay> 个人博客: <http://www.wingjay.com>

♥ 喜欢 (/sign\_in?utm\_source=desktop&utm\_medium=not-signed-in-like-button) | 282




更多分享

被以下专题收入，发现更多相似内容



RN (React Native) (/c/9fd3e700abe7?utm\_source=desktop&utm\_medium=notes-included-collection)



-  iOS Dev... (/c/3233d1a249ca?utm\_source=desktop&utm\_medium=notes-included-collection)
-  @IT·互联网 (/c/V2CqjW?utm\_source=desktop&utm\_medium=notes-included-collection)
-  Swift&ObjC (/c/069807ac166b?utm\_source=desktop&utm\_medium=notes-included-collection)
-  Android... (/c/f358c35b9297?utm\_source=desktop&utm\_medium=notes-included-collection)
-  技术 (/c/847186312793?utm\_source=desktop&utm\_medium=notes-included-collection)
-  集思广益 (/c/ee226afe1061?utm\_source=desktop&utm\_medium=notes-included-collection)

展开更多 ▾

