Introduction to React

Apress®

React导学

[美] Cory Gackenheimer 著
张铮铮 译

- 使用React构建可扩展的、高效的用户界面

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

Introduction
to React

React导学

[美] Cory Gackenheimer 著
张铮铮 译

人民邮电出版社
北京

图书在版编目（CIP）数据

React导学 / (美) 盖肯海默 (Gackenheimer,C.) 著;
张铮铮译. -- 北京 : 人民邮电出版社, 2016.6
ISBN 978-7-115-41943-9

Ⅰ. ①R… Ⅱ. ①盖… ②张… 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(2016)第085042号

版权声明

◆ 著　[美] Cory Gackenheimer
译　张铮铮
责任编辑　陈冀康
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
三河市海波印务有限公司印刷

◆ 开本：800×1000　1/16
印张：12
字数：258 千字　2016 年 6 月第 1 版
印数：1－2 500 册　2016 年 6 月河北第 1 次印刷

著作权合同登记号　图字：01-2016-0776 号

定价：39.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315

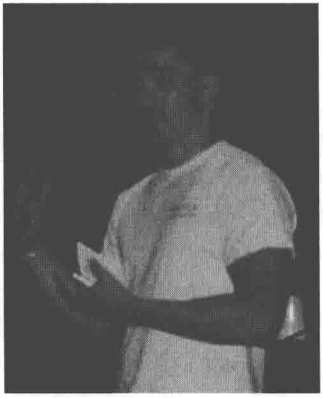
内容提要

React 是一种 JavaScript 框架，起源于 Facebook 公司，用于构建复杂且可维护的用户界面。

本书是介绍 React 的快速实践指南。全书共 6 章，系统地概括了有关 React 的方方面面，详细介绍了 React 的概念、核心、JSX 原理、网页应用的构建、程序架构、Flux 的用法等内容。

本书适合对 React 感兴趣的读者以及想要使用 React 进行前端开发的读者阅读参考。

作者简介



Cory Gackenheimer 是来自美国中西部的软件工程师。他曾在普渡大学学习物理学，并使用超高频示波器的图像分析软件进行图像分析。他的软件经验使他能广泛地利用各种技术，包括 JavaScript。他还是 jQuery Mobile 团队的成员，并经常贡献开源项目。工作之余，他喜欢从事基于 Node.js 的项目，并将 React 吸收到其项目中。

技术审阅者简介

Akshat Paul 是一名开发者，也是《RubyMotion iOS Development Essentials》一书的作者。他在移动和 Web 开发领域经验丰富，并且多年来已经为许多企业和客户交付了应用程序。

另外，Akshat 还经常在各种技术研讨会上演讲。他曾在 RubyConfIndia 和布鲁塞尔的#Inspect-RubyMotion 会议上做了发言，还作为主讲人参加了曼谷和吉隆坡的 Technology Leadership 活动。除了编写代码，Akshat 也会花时间陪伴家人。他喜欢读书，且着迷于健康饮食。

致谢

没有 Louise Corrigan 这位令人惊叹的编辑的邀请就不会有本书，他让我再写一本书的热情真是无法抗拒。我还必须感谢我的家庭，他们容忍我把本该属于他们的时间花费到研究、编写代码和撰写本书上。没有他们的包容，我就没法完成本书。最后谢谢广大读者，你们对于 React 的兴趣让你从这里开启 React 之旅。祝你愉快！

目　录

第1章

什么是 React

它让我有幸见识到了那些备受推崇的不因循守旧者们骨子里的固执。

——爱因斯坦

也许在拿到本书的时候，你已经有了一些 JavaScript 知识。很有可能你大概知道 React 是什么。本章突出 React 作为框架这一层面的关键问题，阐释它所解决的问题，并描述如何利用这些特性以及本书其他信息来改善网页开发实践，创建复杂且可维护的用户界面。

1.1 定义 React

React 是一个 JavaScript 框架。它起初是由 Facebook 的工程师创建，用于解决在开发数据随时间而变的复杂用户界面时的困难。这不是件简单的事，它不仅必须是可维护的，还得是可伸缩的以便服务于 Facebook 这样的规模。React 实际上诞生于 Facebook 的广告组织，他们那儿一直在使用传统客户端的模型-视图-控制器（Model-View-Controller）方法。这些应用程序通常由双向数据绑

定（two-way data binding）和渲染模板组成。React 改变了创建这些应用程序的方式，在 Web 开发中取得了一些大胆的进步。在 2013 年 React 刚发布时，Web 开发社区对其所作所为既有兴趣又似乎感到厌恶。

随着通览全书你会发现，React 挑战了实际上已成为标准的 JavaScript 框架最佳实践。为此，React 引入许多新范式，并改变现状以创建可伸缩和可维护的 JavaScript 应用程序和用户界面。伴随着前端开发思路的转换，React 自带一系列丰富的特性，使得各种技能水平（从刚接触 JavaScript 到具有丰富 Web 开发经验）的开发者都可以创建单页面（single-page）应用程序或友好的用户界面。阅读本书，你会看到这些特性（比如虚拟 DOM、JSX 和 Flux 等概念），并了解它们是如何被用于创建复杂用户界面。

简单来说，你还将看到 Facebook 是如何使用 React Native 持续不断地挑战开发界的。React Native 是一个新的开源库，它利用与 React 的 JavaScript 库相同的原理来创建原生用户界面。通过创建 Native UI 库，React 已经推出了其价值主张——“一次学习，随处可用”。这一模式转变也适用于利用 React 的核心概念做出可维护界面。到目前为止，你可能认为用 React 进行开发时没有什么是它无法做的。但情况并非如此，为了更进一步地了解 React 是什么，你需要先了解 React 不是什么，在本章稍后的部分你会学到。首先，你会理解导致 React 被创造出来的根本问题，以及 React 如何解决这些问题。

1.2　为什么是 React

如前所述，React 与常规的 Web 开发是不同的概念，是对公认的工作流和

最佳实践的一个转变。为什么 Facebook 要躲开这种趋势，转而创建一种全新的 Web 开发过程呢？挑战公认的最佳实践是草率行事吗？还是说创建 React 有普遍的商业理由？

如果看看 React 背后的思考，你将会明白创建它是为了满足 Facebook 面临的一系列特殊技术挑战所产生的特殊需求。这些挑战并非 Facebook 独有，但 Facebook 所做的是立即用一种方法来解决问题，以应对这些挑战。可想而知，这与 Eric Raymond 在他的书（*The Art of Unix Programming*[1]）中总结的 UNIX 哲学类似。书中，Raymond 写了下面这段关于模块化的规则。

要编写复杂软件又不至于一败涂地，唯一的方法就是用定义清晰的接口把若干简单模块组合起来，如此一来，多数问题只会出现在局部，那么还有希望对局部进行改进或优化，而不至于牵动全身。

这恰好就是 React 用来解决复杂的用户界面所需要的方法。在开发 React 时，Facebook 没有创建完全的模型-视图-控制器架构来代替现有框架。没必要去重新发明特殊的轮子和增加创建大规模用户界面问题的复杂性。创建 React 是用来解决奇异问题的。

构建 React 是为了处理用户界面中显示的数据。你可能认为用户界面中显示数据的问题已被解决，你这么想也不算错。不同的是 React 服务于大规模用户界面（Facebook 和 Instagram 规模的界面）中随时间而变的数据。这种界面可以被 React 以外的现有工具创建和解决。事实上，Facebook 在创建 React 前一定已经解决了这些问题。但 Facebook 还是创建了 React，因为它已经有正当的理由，并发现 React 可以被用于解决构建复杂用户界面时遇到的特殊问题。

---

[1] 中文版为《UNIX 编程艺术》。（本书所有脚注均为译者所加。）

1.3　React 解决什么问题

React 不能解决在用户界面设计和前端开发中所遇到的所有问题。React 只解决一系列特殊的问题，简而言之，就是单个问题。正如 Facebook 和 Instagram 所阐明的，React 用于构建数据随时间而变的大规模用户界面。

数据随时间而变的大规模用户界面或许与许多 Web 开发者的工作和业余编码经历都有关联。在现代 Web 开发世界中，你通常会把用户界面的大部分责任转移给浏览器以及 HTML、CSS 和 JavaScript。这类应用程序通常被称为单页面应用程序，常见的请求/响应服务器只能有限地展示浏览器的性能。这是自然的，既然大部分的浏览器都能做复杂的布局和交互，为什么你不这样做呢？

到了周末，项目代码不再是可维护的，问题就来了。你不得不附加额外的代码段，将数据恰当地绑定。有时你不得不重建应用程序，因为接下来的业务需求已经在用户开始一项任务后，不经意地破坏了界面渲染一些交互的方式。这一切导致用户界面脆弱，高度相连，以及不易维护。React 尝试解决所有这些问题。

以你之前看到所提及的客户端模型-视图-控制器架构与模板里的双向数据绑定为例。该应用程序必须包含监听模型的视图，然后视图基于用户交互或模型改变独立地更新它们的表现。在基础应用程序中，这不是一个明显的性能瓶颈，或者更重要的，对开发者的生产力而言，这不是一个明显的瓶颈。该应用程序的规模将不可避免地增长，因为新模型和视图会被加入应用程序。所有的

连接都是通过微妙而杂乱的代码，它们可以指明每个视图及其模型的关系，且很快会变得越来越复杂。在渲染链或遥远的模型深处的某一项正影响着其他项的输出。在大多数情况下所发生的更新可能并不会被开发者所知，因为维护跟踪机制变得逐渐困难。它会让开发和测试代码更为困难，这意味着开发一个方法或新特性并发布它变得更难。如此一来，代码缺乏可预见性，开发时间也飞速猛涨，这正是 React 需着手解决的问题。

首先，React 是一个思想实验。Facebook 认为他们已经写了最初的布局代码来描述应用程序可以也应该看起来像什么，所以为什么不在数据或状态改变应用程序的时候再跑一下启动代码呢？你可能会立刻有些退缩，因为你知道这意味着它们会牺牲性能和用户体验。当你完全替换浏览器中的代码时，你将看到屏幕闪烁和无样式内容出现的一瞬间，这只是显得效率低下。Facebook 知道这一点，而且注意到他们所创建的（在数据改变时代替状态的机制）实际上在某种程度上是有作用的。然后 Facebook 决定，只要替换机制可以被优化，那就有解决方案了。这就是 React 如何因为一组特定问题的解决方案而诞生的。

1.4 React 不只是另一个框架

大多数情况下，你要学某样东西，首先需要认识到你要学的是什么。就 React 来说，了解哪些概念不属于 React 框架是有帮助的。这将有助于你理解你所学的哪些标准实践需要忘掉，或者至少需要放一边，以彻底理解新框架的概念，比如 React。那么是什么使 React 与众不同，以及为什么它如此重要呢？

许多人认为，React 相比于其他框架是同等级的完整 JavaScript 框架，比

如 Backbone、Knockout.js、AngularJS、Ember、CanJS 和 Dojo，或大多数其他现有 MVC 框架。图 1-1 显示了经典 MVC 框架的示例。

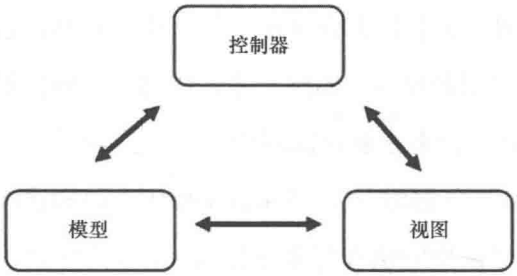


图 1-1　基本 MVC 架构

图 1-1 展示了模型-视图-控制器架构中每个组件的基本要素。模型处理应用程序的状态，并将状态改变事件发送给视图。视图是面向用户的外观和对终端用户的交互界面。视图可以将事件发送给控制器，有时候也发送给模型。控制器是事件主要的分派器，事件会被发送给模型，以更新状态，然后视图来更新表现。你可能会注意到这是一个常见的 MVC 架构的代表，在实际中有很多变体和定制化实现，因此不存在单一的 MVC 架构。这里不是声明 MVC 架构看起来像什么，而是指出 React 不是什么。

用这种 MVC 架构来评价 React 是什么或想成为什么，实际上不公平。这是因为 React 是那些现有框架中的特例。React 处于其最简形式，仅为 MVC、MVVM 或者 MV*框架中的视图。如你在之前所见，React 是一种描述应用程序用户界面的方法，是一种在数据发生变化时随时更改用户界面的机制。React 由描述界面的声明性组件组成。在构建应用程序时，React 没有使用可观察的数据绑定。React 也是易于维护的，因为你可以用你创造的组件，并组合它们，在任何你期望的时候自定义组件，因为它可以扩展。促使 React 出现的那些理由，使得 React 可以比其他框架更好地扩展。在创建 React 界