Web开发经典丛书

使用React、JSX、Redux和GraphQL开发Web Apps

快速上手 React 编程

John Sonmez
作序推荐本书

[美] 阿扎•马尔丹 (Azat Mardan) 著
郭美青 郭 松 唐金州 译

MANNING

清华大学出版社

Web 开发经典丛书

快速上手 React 编程

[美] 阿扎•马尔丹（Azat Mardan） 著
郭美青　郭　松　唐金州　译

清华大学出版社

北　京

Azat Mardan
React Quickly
EISBN: 978-1-61729-334-4

北京市版权局著作权合同登记号　图字：01-2017-7950

图书在版编目(CIP)数据

快速上手 React 编程 / (美)阿扎•马尔丹(Azat Mardan)　著；郭美青，郭松，唐金州　译. —北京：清华大学出版社，2018
(Web 开发经典丛书)
书名原文：React Quickly
ISBN 978-7-302-50247-0

Ⅰ. ①O…　②快…　Ⅱ. ①阿…　②郭…　③郭…　④唐…　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(2018)第 103232 号

责任编辑：王　军　李维杰
装帧设计：思创景点
责任校对：孔祥峰
责任印制：刘海龙

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：清华大学印刷厂
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：28.75　　**字　　数**：700 千字
版　　次：2018 年 6 月第 1 版　　**印　　次**：2018 年 6 月第 1 次印刷
印　　数：1～3500
定　　价：88.00 元

产品编号：075962-01

本书赞誉

“对于任何想要获得 React 入门指导，并了解其周边生态系统(包括工具、概念和库)的人来说，本书提供了一种一站式服务。跟随 Azat 的演练，完成给定的项目，你将很快理解 React、Redux、GraphQL、Webpack 和 Jest，以及如何让它们运行起来。”

——Peter Cooper，*JavaScript Weekly* 杂志编辑

“本书通过 React 向读者传授在构建现代 Web 应用的过程中最具价值和值得关注的概念，包括 GraphQL、Webpack 和服务器端渲染。阅读之后，你应该有足够的信心能够使用 React 创建一个生产级的 Web 应用。”

——Stan Bershadskiy，*React Native Cookbook* 一书的作者

“Azat 是编程领域最具权威的声音之一。本书深入研究了 React 的基础和架构，并远远超越了基础。对于任何开发者来说，它都是必读的!”

——Erik Hanchett，*Ember.js Cookbook* 一书的作者

“这本书很简单。它使用非常基础的语言让你逐步了解每一个概念。”

——Israel Morales，SavvyCard 前端开发者和 Web 设计师

“简单的语言配以简单的逻辑示例，让你起步并快速运行程序，本书涵盖了任何 React 新手在开始编写应用时都会遇到的主要问题。作者的幽默感会让你一直忙到最后。感谢 Azat 花时间与我们分享他的 React 之旅。”

——Suhas Deshpande，Capital One 软件工程师

“本书是加速学习 React 的好资源，非常透彻和中肯。我将用它作为开发下一个应用的参考。”

—Nathan Bailey，SpringboardAuto.com 全栈开发者

“Azat 非常擅长他所做的事情——教人如何写代码。本书包含基础知识和实例，能让你快速开始使用 React。”

—Shu Liu，IT 顾问

“自 2013 年被 Facebook 开源以来，React 已经迅速成为一个被广泛采用的 JS 库，并且

是 GitHub 上最著名的项目之一。在他的新书《快速上手 React 编程》中，Azat Mardan 以他典型的简明风格，把你需要了解的关于 React 生态系统的一切都编排好了，以便快速构建高性能的 SPA 应用。仅仅关于 React 状态和同构 JavaScript 的章，就值得你购买和阅读。”

—Prakash Sarma，New Star Online

“本书将通过传达清晰的基础知识来让你慢慢接纳 React，它将让你构建应用并彻底接受使用 React 带来的好处。”

—Allan Von Schenkel，FoundHuman 技术与战略副总裁

“本书以易于理解的方式涵盖了 React 的所有重要方面。和 Azat 的所有作品一样，本书清晰而简洁，并且包含快速提高生产力所需的内容。如果你有兴趣将 React 加入自己的技能集，我建议从这里开始。”

—Bruno Watt，hypermedia.tech 咨询架构师

“这是一本关于使用 React 进行全栈 Web 开发的非常全面的书，不仅涵盖 React 本身，还包含周边的生态系统。我总是对 React 服务端渲染感到困惑，Azat 的这本书终于帮我理解了它。如果你是 React 新手并想要真正掌握它，看这本书就足够了。”

—Richard Kho，Capital One 软件工程师

译 者 序

在本书翻译之初，React 社区刚刚经历了两个重量级的事件。一件是前后持续一年有余的 Facebook 附加专利条款事件，后因“React 能不能继续商用”的问题在国内被炒得沸沸扬扬，但随着事件持续发酵，最终迎来了让开发者皆大欢喜的局面；另一件是 React 团队发布了最新的基于 Fiber 架构的 v16.0.0 正式版，带来了众多优秀特性和性能提升。一方面我们看到了 React 无与伦比的业界影响力，甚至连 Facebook 都不惜以得罪开源社区的方式，希望从 React 身上获得商业利益保护的能力；另一方面我们又看到一支孜孜不倦、勇于探索、不断进取的行业先锋团队，当大部分人还在纠结该使用 React、Vue 还是 Angular 2 的时候，React 团队再一次用 Fiber 架构突破浏览器自身的局限，为 React 的发展带来更大的想象空间。React 无疑是当下最热门、最具前景的前端技术之一。

当下，我们正处在“JavaScript 文艺复兴时代”。各种标准、思想、框架、类库百花齐放，各种工具、概念、术语层出不穷。条条大路通罗马，要实现一个前端应用，解决方案除了当年那个永恒的 jQuery 以外，可谓数不胜数，涉及的专有名词更是目不暇接。这是件令人兴奋的事情，因为这意味着前端生态在不断壮大，JavaScript 的能力也在日益增强。但对于前端从业者而言，却是不小的挑战。

依稀记得刚接触 React 时，面对扑面而来的新鲜术语的那种困惑，就好像看见散落一地的珍珠，却不知该如何捡起串成美丽的项链一样。Azat 无疑非常擅长这项工作，他把自己在一线教授 React 的课程、经验、资源以及学员的反馈重新梳理、优化、整合，按照由浅入深、由表及里的编排方式，不仅关注核心，也注重生态，同时配上丰富的实战案例和架构思想解读，为我们带来了这本学习 React 不容错过的《快速上手 React 编程》。相信这本书一定会为读者带来醍醐灌顶般的阅读体验，因为该书不仅仅教你如何使用 React，还告诉你 React 技术体系中每一项设计的原因和思考。希望读者能从中受益，享受学习和使用 React 带来的快乐，这也是译者莫大的荣幸。

本书由郭美青、郭松、唐金州合译完成。特别感谢梁宵的引荐，让我们得以共同合作参与本书的翻译，在整个翻译过程中，他高效的组织、协调能力和审校的专业让我们受益匪浅。感谢我们的家人，你们的支持和鼓励让我们感到无比的温暖。感谢清华大学出版社李阳老师的悉心指导以及在翻译过程中给予的极大理解和帮助。

我们在本书翻译过程中力求做到行文流畅、风格一致，希望能给读者带来愉悦的阅读体验，并尽力修正一些原书中的小错误，但鉴于自身水平有限，疏漏在所难免，敬请广大读者批评指正。

最后，希望本书能帮助业界同仁系统地掌握 React 并使用 React 打造出高性能、更易于维护的 Web 产品。

译者

序　言

认真地讲，我一直希望 JavaScript 可以凋零、磨灭。

这并不是说我完全不喜欢 JavaScript——多年以来，它已经改进了不少；而是因为我对复杂的极度厌恶——以至于我将我的博客和公司起名为 Simple Programmer。我的口号一直是：“让复杂变得简单。”

让复杂变得简单并不容易。它需要一套特殊技能。你必须能够理解复杂，并把它理解得很好，这样你才能够从中提炼出核心——因为任何事情在核心上都是简单的。这正是 Azat 在本书中所做的。

现在，我承认 Azat 提供了一点帮助。你能看出，我个人喜欢 React 的原因之一就是它很简单。它被设计得很简单，用于处理越来越复杂的 JavaScript 框架，并通过回归基础(即原始的 JavaScript)来降低复杂性(至少在大多数情况下是这样的。React 的确有一种可以编译成 JavaScript 的 JSX 语言，但是我会让 Azat 介绍给你)。

重点是，尽管我喜欢 Angular、Backbone，以及一些其他的 JavaScript 框架，因为它们能够帮助 Web 开发者更容易地创建异步 Web 应用和单页面应用，但这同时也增加了大量的复杂性。使用模板并理解这些框架的语法和微妙之处能提高生产率，但它们把复杂性从后端转移到了前端。React 一开始就摒弃了模板，而是提供了一种使用 JavaScript 将基于组件的架构应用于 UI 的方式。我喜欢这样。它很简单。但即使最简单的事情也很难解释——或者更糟糕的是，被缺乏这种能力的老师变复杂了。

这正是 Azat 的强项。他知道如何教学，如何简化。他通过解释 React 开始本书，对比了你可能已经知道的 Angular。即使你不了解 Angular，他对 React 的解释也能快速帮助你理解其基本原理和用途。然后，Azat 很快示范了如何创建一个基本的 React 应用，这样你就能了解并亲自实践了。在那之后，他会通过任何人都很容易掌握的真实示例，带你去了解你需要知道的 20%，以便完成你将要在 React 中做的剩余 80%。最后——也是我最喜欢的部分——里面包含了丰富的例子和项目。学习的最佳方式绝对是实践，而 Azat 会使用 React 带你去创建 6 个非凡的项目。

为了保持我简单的风格，让我最后说一句：《快速上手 React 编程》提供了我所知道的学习 React 的最佳方法。

John Sonmez

Soft Skills 一书的作者(http://amzn.to/2hFHXAu)

Simple Programmer 创始人(https://simpleprogrammer.com)

作者简介

我已经出版了超过14本书和17门在线课程(https://node.university)，它们中的大多数存储在云端，涉及React、JavaScript和Node.js。在关注Node之前，我也使用过其他语言编程(Java、C、Perl、PHP、Ruby)，几乎从高中开始编程(十多年前)，并且绝对超过了规定的一万小时。

现在，我是美国十大银行之一的技术研究员，它也是一家财富500强公司：Capital One Financial Corporation，位于美丽的旧金山。在那之前，我曾为小型初创公司、大型企业，甚至美国联邦政府工作过，编写桌面、Web和移动应用，从事教学、开发协调和项目管理。

我不想占用你太多的时间介绍自己，你可以在我的博客(http://webapplog.com/about)和社交媒体(www.linkedin.com/in/azatm)上了解我的更多信息。相反，我想把关于这本书的经历写下来。

2011年，我搬到了阳光明媚的加州，开始通过企业加速器创业(如果你对此好奇，它就是500家初创公司)，开始使用现代的JavaScript。我学会了使用Backbone为公司创建一些应用，并对此印象深刻。该框架极大地改进了我前几年构建的单页面应用的代码组织。它有路由和模型。这很棒！

我在DocuSign(去Google搜索一下e-signatures，它的市场占有率达到70%)担任软件工程团队领导时，又有机会看到了Backbone和同构JavaScript的惊人能力。我们重构了一个有7年历史的ASP.NET Web应用，每一个小版本的发布都需要四周的时间，而使用时髦的Backbone-Node-CoffeeScript-Express应用不仅具有非常好的用户体验，而且发布版本只需要一到两周的时间。设计团队在可用性方面做得很好。毋庸置疑，有大量具有不同程度交互的UI视图。

最终的应用是同构的，甚至在此之前还不存在该术语。我们使用服务器上的Backbone模型从API获取数据并将其缓存。我们在浏览器和服务器上使用相同的Jade模板。

这是一个有趣的项目，它让我更加相信使用一种语言横跨整个技术栈的力量。精通C#和前端JavaScript(大部分是jQuery)的老款应用开发者，如果花点时间突击一下(一个发布周期，通常是一到两周)，就会爱上结构清晰的CoffeeScript、Backbone的组织结构以及Node的速度(开发和运行速度)。

我在Web开发领域十几年的经验向我揭示了好的、坏的以及丑陋的(大部分是丑陋的)前端开发。然而柳暗花明，因为自从切换到React，我就越来越喜欢它了。

如果你想收到更新、新闻和小贴士，可以在网上通过以下方式联系我，订阅、加好友、

悄悄关注均可：

- Twitter—https://twitter.com/azat_co
- 网站—http://azat.co
- LinkedIn—http://linkedin.com/in/azatm
- 专业博客—http://webapplog.com
- 出版物—http://webapplog.com/books

关于面对面研讨会和课程，请访问 http://NodeProgram.com 或 https://Node.University，或通过 https://webapplog.com/azat 给我发消息。

致　　谢

我要感谢互联网、全世界以及人类的聪明才智，让我们相信心灵感应是可能的。不必开口，就可以通过 Twitter、Facebook 和 Instagram 等社交媒体与全球数百万人分享自己的想法。

我对我的老师心存感激，无论是在中学还是在大学，不经意间，我从书本和耳濡目染的学习中领悟到他们的智慧。

史蒂芬·金说过：“写作是人类做的事情，编辑是上帝做的事情。”因此，我要无数次感谢本书的编辑，更要感谢那些读者，他们在本书中不得不面对不可避免的打字错误和 bug。这是我的第 14 本书，我知道无论怎样都会产生这些问题。

感谢 Manning 出版社的工作人员，他们使本书付梓成为可能：发行人 Marjan Bace 以及所有编辑制作团队，包括 Janet Vail、Kevin Sullivan、Tiffany Taylor、Katie Tennant、Gordan Salinovic、Dan Maharry，还有很多其他的幕后工作者。

我不知如何表达对 Ivan Martinovic 领导的超凡技术审校人团队的感谢：James Anaipakos、Dane Balia、Art Bergquist、Joel Goldfinger、Peter Hampton、Luis Matthew Heck、Ruben J. Leon、Gerald Mack、Kamal Raj 和 Lucas Tettamanti。他们的贡献包括捕获术语中的技术失误、错误、拼写错误，并提供专题建议。每个人都经历了审校过程，论坛主题中的每一条反馈塑造了本书的手稿。

在技术方面，特别感谢 Anto Aravinth，他是本书的技术编辑；还要感谢 German Frigerio，他是本书的技术校对。他们正是我所期望的最好的技术编辑。

非常感谢 John Sonmez 为本书作序，他闻名于 Pluralsight、Manning 出版社以及 SimpleProgrammer.com。感谢 Peter Cooper、Erik Hanchett 和 Stan Bershadskiy 所做的审校工作，你们赋予本书额外的可信度。没有听说过 John、Peter、Erik 或 Stan 的读者应该关注他们在软件工程领域所做的工作。

最后，感谢所有 MEAP 读者的反馈。基于你们的评论，我修订了本书，虽然使本书出版延迟了一年，但却造就了当下关于 React 的一本好书。

前　言

那是 2008 年，银行纷纷倒闭。我在联邦存款保险公司(Federal Deposit Insurance Corporation，FDIC)工作，主要任务是偿还倒闭、失败和破产的银行储户。我承认，就职业安全感而言，我的工作等同于在雷曼兄弟上班或在泰坦尼克号上卖票。但是，当我所在的部门还没有开始最终的预算削减时，我还有机会开发一款名为 EDIE 的应用。这款应用由于一个简单的原因而变得非常流行：人们急于想知道他们的存款中有多少由美国联邦政府提供保险，而 EDIE 能够估算这一数额。

但是存在一个问题：人们不喜欢把他们的私人账户告诉政府。为了保护他们的隐私，这款应用完全在前端完成：JavaScript、HTML 和 CSS，没有任何后端技术。如此，FDIC 就不会收集任何个人财产信息。

这款应用经历了数十次迭代，留下了一团混乱的意大利面条式代码。开发人员来去匆匆，没有留下任何文档，也没有任何符合逻辑的简单算法。这就像没有地图就要去乘坐纽约市地铁一样。有很多函数可以调用其他函数、奇怪的数据结构以及更多的函数。在现代术语中，这款应用采用纯用户界面(User Interface，UI)，因为没有后端。

我多么希望那时能有 React。React 带来了愉悦。它是一种全新的思考方式，也是一种全新的开发方式。将核心功能放在一处，而不是将其分解为 HTML 和 JS，这种简单是一种解放。它重新点燃了我对前端开发的热情。

React 是开发 UI 组件的一种新方法。它是新一代的表现层库。与模型和路由库一起配合，React 可以在 Web 和移动技术栈中取代 Angular、Backbone 或 Ember。这就是我写这本书的原因。我从来不喜欢 Angular：它太复杂和大一统了。模板引擎是特定领域的，甚至不再是 JavaScript；它是另一种语言。我使用过 Backbone，喜欢它的简单和 DIY 方式。Backbone 是成熟的，它更像是你自己框架的基石，而不是一个全面的、大一统的框架。Backbone 的问题是模型与视图之间不断增加的交互复杂性：多个视图更新多种模型，有的会更新其他视图，有的会触发模型上的事件。

举办 React 在线课程(http://mng.bz/XgkO)，以及参加各种会议和活动的个人经验已经显示，开发者渴望一种更好的开发 UI 的方法。大多数商业价值存在于 UI，而后端负责提供商品。在海湾地区，也是我居住和工作的地方，大多数软件工程师的职位空缺是前端或(一个时髦的头衔)通才/全栈开发者。只有少数像谷歌、亚马逊和 Capital One 这样的大公司，对数据科学家和后端工程师的需求依然强劲。

想要确保职业安全感，或找到一份好工作，最好的方法就是成为一名多面手。要做到这一点，最快的方法是使用同构的、可扩展的、对开发者友好的库，就像前端的 React 搭配后端的 Node.js，以防止需要处理服务器端代码。

对于移动开发者来说，HTML5 是两三年以前的一个肮脏词汇。Facebook 放弃了 HTML5 应用，转而支持更高效的原生(native)实现。但是这种不利观点正在迅速改变。通过 React Native，可以渲染移动应用：可以保留 UI 组件，但在不同的环境下调整它们，这是另一种支持学习 React 的观点。

编程可以是创造性的。不要陷入乏味的任务、复杂的事情以及伪关注点分离。砍掉所有不必要的垃圾，通过 React 提供的简化的模块之美、基于组件的 UI 释放你的创造力。为同构的 JavaScript 引入一些 Node，你将达到禅定的境界。

祝你阅读愉快，并让我知道你对这本书的评价，请在 Amazon.com(http://amzn.to/2gPxv9Q)上留言。

学习建议

本书旨在解决前端开发者的烦恼，使他们的生活更有意义、更加快乐，并通过介绍 React 帮助他们赚到更多的钱——这些会以一种快速的方式进行。本书是十几个人工作一年半的成果。至少，本书会开拓你的思维，让你了解一些不寻常的概念，如 JSX、单向数据流和声明式编程。

路线图

本书分为两部分：“React 核心”(第 1 至第 11 章)和“React 生态”(第 12 至第 20 章)。每一章都会为描述性的文字配以恰当的代码示例和图表。每一章也都有一段可选的介绍视频，帮助你决定是否需要阅读该章。各章是以独立的方式编写的，这意味着即便不按顺序阅读，也不会带来麻烦——尽管建议按顺序阅读。每章的结尾是一个小测验，以强化你对该章内容的记忆，最后还有小结。

每一部分都以一系列更大的项目结尾，这些项目将给你带来更多的 React 经验，通过前几章介绍的概念和知识构建应用，巩固你对知识的理解。这些项目以可选的截屏视频作为补充来强化你的学习，向你展示动态的过程，如创建文件和安装依赖(Web 开发包含很多组成部分！)。这些项目是书中不可或缺的一部分——不要跳过它们。建议你自己键入每一行代码，并避免复制和粘贴。研究表明：打字和写作能提高学习效率。

本书以 5 个附录为结尾，它们提供了补充材料。在开始阅读之前，请连同目录一起把它们看一遍。

本书的网站是 www.manning.com/books/react-quickly 和 http://reactquickly.co。如果需要最新的信息，很可能会在那里找到。

源代码可以在 Manning 出版社的网站(www.manning.com/books/react-quickly)和 GitHub(https://github.com/azat-co/react-quickly)上找到。查看“源代码”部分可获得更多详情。读者也可扫描本书封底的二维码来下载源代码和每章的介绍视频。书中已展示完整的代码清单——这比跳转到 GitHub 或代码编辑器去查看文件方便得多。

本书适合的读者(必读！)

本书适合 Web 和移动开发者以及具备两到三年工作经验的软件工程师阅读，他们想开始学习并使用 React，用于 Web 或移动开发。基本上，本书适合那些对开发者工具的快捷

键了然于心的人(比如 Mac 上的 Cmd+Opt+J 或 Cmd+Opt+I)。本书的目标读者是知晓并熟悉以下概念的人：

- 单页面应用(Single Page Application，SPA)
- RESTful 服务和 API 架构
- JavaScript，尤其包括闭包、作用域，以及字符串和数组方法
- HTML、HTML5，以及它们的元素和属性
- CSS 及其样式，以及 JavaScript 选择器

有 jQuery、Angular、Ember、Backbone 或其他 MVC 类框架的使用经验是一个加分项，因为你可以用 React 的方式来对比它们。但这是没有必要的，在某种程度上甚至可能是有害的，因为你需要忘掉某些模式。React 并不完全是 MVC。

你将使用命令行工具，如果害怕它们，那么这是对抗命令行/终端/命令提示符恐惧的最佳时机。一般来说，CLI 比它们的可视化(GUI)版本更加强大和通用(比如 Git 的命令行版及桌面版，后者把我搞糊涂了)。

熟悉 Node.js 会让你比从未听说过 Node.js、npm、Browserify、CommonJS、Gulp 或 Express.js 的人学得更快。我写过几本关于 Node.js 的书，对于那些想要重温它的人，最流行的就是 *Practical Node.js*(http://practicalnodebook.com)。或者，你也可以去网上寻找免费的 NodeSchool 课程(http://nodeschool.io)(免费并不总是意味着更糟)。

本书未包含的内容(也请必读！)

本书并不是一本全面的 Web 或移动开发指南。本书假设你已经了解这些知识。如果你想学习基本的编程概念或 JavaScript 基础知识，有很多关于这些主题的好书。我想到的有：Kyle Simpson 的 *You Don't Know Js*(https://github.com/getify/You-Dont-Know-JS)、*Secrets of the JavaScript Ninja, Second Edition*(www.manning.com/books/secrets-of-the-javascript-ninja-second-edition)，以及 Marijn Haverbeke 的 *Eloquent JavaScript*(http://eloquentjavascript.net)。所以，我没必要在本书中重复已有的内容。

如何使用本书

首先，你应该阅读这本书。这不是开玩笑。大多数人把书买回来，但从不去读。阅读这本书吧，并逐章去完成那些项目。

每一章都涉及一个或一系列主题，它们互为基础。因此，建议你从头到尾阅读这本书，然后将个别章作为参考。但正如之前所说，也可以不按顺序阅读个别章，因为各章的项目是独立的。

有许多外部资源的链接。它们中的大多数是可选的，并提供了关于该主题的额外详情。因此，建议在电脑旁阅读本书，以便在提到它们时可以打开链接。

有时你会看到带有奇怪缩进的代码，比如：

```
 document.getElementById('end-of-time').play()
}
```

这表示正在注解一大块代码并将其分解成块。这一块属于一个从位置 0 开始的更大的代码清单，这一小块代码不能自己运行。

其他时候，代码块没有缩进。这种情况下，可以假设代码片段是完整的：

```
ReactDOM.render(
<Content/>,
  document.getElementById('content')
)
```

如果看到一个美元符号($)，这表明是一条终端/命令提示符命令。例如：

```
$ npm install -g babel@5.8.34
```

在使用本书的时候，最重要的是要知道并记住：你必须乐在其中。如果不好玩，那就不是 JavaScript！

源代码

书中的所有代码可从 www.manning.com/books/react-quickly 和 https://github.com/azat-co/react-quickly 下载，也可通过扫描封底二维码获得。请遵循文件夹命名约定 chNN，此处 NN 是以 0 开始的各章编号(例如，ch02 文件夹代表里面是第 2 章的代码)。GitHub 仓库中的源代码会不断更新，包括补丁、bug 修复，甚至可能是新的版本和风格(ES2020?)。访问网址 http://reactquickly.co/demos，可以得到各章的一些示例。

勘误

书中肯定会有打字错误。是的，虽然编辑们都很负责——他们都是 Manning 出版社的专业人士。但是，我仍对你能找到那个错误表示感激。请不要给我发送 bug 或打字错误。取而代之，你可以在本书的论坛(https://forums.manning.com/forums/react-quickly)上报告，或者在 https://github.com/azat-co/react-quickly/ issues 上创建一个 GitHub issue。通过这种方式，其他人也可以受益于你的发现。

另外，请不要给我发邮件讨论技术问题或勘误。请把它们发在本书的论坛、GitHub 的页面(https://github.com/azat-co/react-quickly)或 Stack Overflow 上。其他人也许能比我更快(而且更好)地帮助你。

本书论坛

购买《快速上手 React 编程》后，可以免费访问由 Manning 出版社运营的私有 Web 论坛，在这里可以对本书发表评论，提出技术问题，并获得作者和其他用户的帮助。该论坛

的网址是 https://forums.manning.com/forums/react-quickly。也可以在 https://forums.manning.com/forums/about 上了解到更多关于 Manning 论坛和行为准则的信息。

Manning 出版社对读者的承诺是提供一个场所，在这里各位读者之间，读者与作者之间，可以碰撞出思想的火花。我们不承诺作者现身的次数，他们对该论坛的贡献是自愿的(并且是无酬劳的)。建议你尽量向作者问一些具有挑战性的问题，以激发他的兴趣！

目　录

第Ⅰ部分　React 基础

第 I 部分

React 基础

你好！我是 Azat Mardan，我即将带你进入一段美妙的 React 世界之旅。这将使前端开发更加愉悦，代码更易于编写和维护，用户也会对应用的速度感到欣喜。React 改变了 Web 开发的游戏规则：React 社区开创了许多方法、术语和设计模式，并且其他库也都追随了 React 的脚步。

我已经在线上直播和个人研讨会上教授这些内容 20 余次，听众是数以百计的软件工程师，他们具有不同的背景和资历。因此，这些内容是经过实践检验过的：它们是我的 React 基础课程经过提炼后的、最有效的书面版本。这些章对于你熟悉 React 术语至关重要。

第 1 至第 11 章是多人近两年来工作的成果，里面介绍的主题层层递进，需要快速阅读。阅读这些章的最佳方法是从第 1 章开始，按照顺序进行。每章都包含一段视频，第 1 至第 8 章的最后还有一个测验，第 9 至第 11 章是具体项目，包含需要你自主完成开发的作业。

总而言之，本书的这一部分为 React 概念、模式和特性构建了坚实的基础。到了国外，不通过学习就可以理解它们的语言吗？显然不行，这就是为什么在尝试构建复杂应用之前，必须先学习 React“语言”的原因。因此，学习这些 React 基本概念至关重要。学习 React 语言正是接下来的 11 章要完成的工作。

让我们开始使用 React，并学会流利的 React 语言吧！

第 1 章

初识 React

本章内容：

- 理解什么是 React
- 使用 React 解决问题
- 把 React 添加到 Web 应用中
- 编写第一个 React 应用：Hello World

2000 年初，在我开始从事 Web 开发工作时，所需要的只是一些 HTML 和诸如 Perl 或 PHP 这样的服务器端语言。那些使用 alert( )弹出信息的美好时光只是为了调试前端代码。事实上，随着互联网的发展，构建网站的复杂度已经急剧增加。网站已经成为具有复杂用户界面、业务逻辑和数据层的 Web 应用，并且需要随时间进行修改和更新，且通常是实时的。

许多 JavaScript 模板库已被创建用来尝试解决处理复杂用户界面(UI)的问题。但是，它们仍需要开发人员坚持旧的关注点分离原则——分离样式(CSS)、数据和结构(HTML)以及动态交互(JavaScript)，而它们并不能满足现代需求。(还记得术语 DHTML 吗？)

相比之下，React 提供了一种简化前端开发的新方法。React 是一个功能强大的 UI 库，提供了许多大公司(如 Facebook、Netflix 和 Airbnb)已经采纳和认定的替代方法。React 允许使用 JavaScript 创建可重用的 UI 组件，而不是为 UI 定义一次性的模板，可以在网站中反复使用这些组件。

还需要验证码控制器或日期选择器吗？使用 React 定义一个<Captcha/>或<DatePicker/>组件，就可将它们添加到表单中：这就是一个简单的插件组件，具备与后端通信的所有功能和逻辑。当用户输入四个或更多个字母时，是否需要一个自动完成框来异步查询数据库？定义一个<Autocomplete charNum="4" />组件就可以进行异步查询。还可以选择是否存在一个文本框 UI 或者没有 UI，此时可以使用另一个自定义表单元素，也许是<Autocomplete textbox="…" />。

这种做法并不新奇。“创建可组合的 UI”的方法已经存在很长时间了，但 React 是第一个使用无模板的纯 JavaScript 来实现的。这种方法已被证明易于维护、重用和扩展。

React 是一个很好的 UI 库，它应该成为 Web 前端工具包的一部分；但它并不是所有 Web 前端开发的完整解决方案。在本章中，我们将介绍在应用中使用 React 的优缺点，以及如何将其适配到现有的 Web 开发技术栈中。

本书的第Ⅰ部分重点介绍 React 的主要概念和特性，第Ⅱ部分着重讨论与 React 相关的库，以构建更复杂的前端应用(也就是 React 技术栈或 React 全家桶)。每部分都展示使用 React 和最受欢迎的库进行的新系统(绿地)和遗留系统(棕地)的开发[1]，所以你可以了解如何在真实场景中使用它。

各章的视频和源代码

每个人的学习方法都不同。有些人喜欢文字和视频，有些人更喜欢通过面授教学学习。本书的每一章都包含一段简短的视频，在不到 5 分钟的时间内就可以解释该章的要点。观看视频是可选的，如果喜欢视频方式或需要复习，它们可以为你提供摘要。观看每段视频后，可以决定是否需要阅读该章，或是跳到下一章。

本章示例的源代码位于 www.manning.com/books/reactquickly 和 https://github.com/azat-co/react-quickly/tree/master/ch01(在 Github 仓库 https://github.com/azat-co/react-quickly 的 ch01 文件夹中)。可以在 http://reactquickly.co/demos 上找到一些示例。

1.1 什么是 React

要合理解释 React，首先需要给它下个定义。那么，什么是 React 呢？它是一个 UI 组件库。这些 UI 组件通过 React 使用 JavaScript 而不是一种特殊的模板语言创建。这种方法被称为“创建可组合的 UI”，它是 React 的理念基础。

React UI 组件是高度自包含、有特定关注点的功能单元。例如：可能有日期选择器、验证码、地址和邮政编码组件。这些组件具有视觉展示和动态逻辑。某些组件甚至可以自己与服务器通信：例如，自动完成组件可能会从服务器提取自动完成列表。

用户界面

广义上，用户界面[2]是促进计算机和人类之间交互的一切。想想一张穿孔卡片(早期的为计算机输入指令用的一种纸质卡片，这个术语源自 Herman Hollerith，他在 1890 年为美国人口局设计了电子制表系统，采用了穿孔卡片)或鼠标，它们都是 UI。当进入软件时代时，工程师们谈论的图形用户界面(Graphical User Interface，GUI)是早期的个人电脑(如 Mac 和 PC)的先驱。GUI 包括菜单、文本、图标、图片、边框和其他元素。Web 元素只是 GUI 的一小部分，它们位于浏览器中，但 Windows、OS X 和其他操作系统中也有用于桌面应用的元素。当在本书中提到 UI 时，意指 Web GUI。

1 “棕地”指的是包含遗留代码的已有系统，“绿地”指的是没有遗留代码的新系统，参阅 https://en.wikipedia.org/wiki/Brownfield_(software_development)

2 https://en.wikipedia.org/wiki/User_interface

不要将基于组件的架构(Component-Based Architecture，CBA)和 Web 组件搞混淆，Web 组件只是在 React 之前存在的最新 CBA 实现之一。这种架构通常被认为比单个庞大的 UI 更容易重用、维护和扩展。React 带来的好处是可以使用纯 JavaScript(无模板)和一种新的视角来看待组件的组合。

1.2　React 解决的问题

React 解决了什么问题？放眼近几年的 Web 开发，你会注意到在前端应用中构建和管理复杂 Web UI 存在的问题，React 主要就是为了解决这些问题而生。试想一下，像 Facebook 这样的大型网络应用，开发此类应用时最痛苦的任务之一就是管理视图对数据变更的响应。

浏览 React 官网以便了解这个问题的更多信息：“我们构建 React 来解决如下问题：构建包含随时间变化的数据的大型应用”[3]。这很有趣！我们还可以查看 React 的历史来了解更多信息。React Podcast[4]上的一个讨论提到，React 的创始人 Jordan Walke 正在 Facebook 上解决一个问题：一个自动完成字段，有多个数据源对其进行更新，这些数据都是从后端异步获取的。确定在哪里插入新行以便重用 DOM 元素变得越来越复杂。Walke 决定每次重新生成这个字段的 UI 展示(DOM 元素)。这个解决方案因其简洁性而变得非常优雅：UI 作为函数，数据作为参数并调用这种函数，并且使渲染视图结果变得可预测。

之后，事实证明在内存中生成元素非常快，实际瓶颈出现在渲染到 DOM 中时。但是 React 团队提出了一种可以避免非必要 DOM 渲染的算法。这使得 React 非常快速(在性能方面)。React 的卓越性能表现，以及对开发人员友好的基于组件的架构是非常成功的组合。React 的这些相关内容和其他优点将在下一节中介绍。

React 解决了 Facebook 的初始问题，许多大公司都认同这种做法。React 采用的方案非常稳固，而且人气逐月攀升。React 源自 Facebook[5]，但现在不仅被 Facebook 使用，Instagram、PayPal、Uber、Sberbank、Asana[6]、Khan Academy[7]、HipChat[8]、Flipboard[9]以及 Atom[10]都在使用 React。[11]大多数这些应用最初使用的是其他技术(通常是带有模板引擎的 Angular 或 Backbone)，但现在都非常乐于切换到 React。

3　React官网，“Why React?”，2016/03/24，http://bit.ly/2mdCJKM

4　React Podcast,“8. React, GraphQL, Immutable & Bow-Ties with Special Guest Lee Byron”，2015/12/31，http://mng.bz/W1X6

5　“Introduction to React.js”，2013/07/08, http://mng.bz/86XF

6　Malcolm Handley和Phips Peter,“Why Asana Is Switching to TypeScript”，Asana Blog, 2014/11/14, http://mng.bz/zXKo

7　Joel Burget,“Backbone to React”，http://mng.bz/WGEQ

8　Rich Manalang,“Rebuilding HipChat with React.js”*Atlassian Developer*，2015/02/10, http://mng.bz/r0w6

9　Michael Johnston, “60 FPS on the Mobile Web”Flipboard, 2015/02/10, http://mng.bz/N5F0

10　Nathan Sobo,“Moving Atom to React”Atom, 2014/07/02, http://mng.bz/K94N

11　另请参阅JavaScript使用统计信息，http://libscore.com/#React

1.3 使用 React 的好处

所有新的库或框架都会声称相比它的前辈在某些方面做得更好。一开始，我们拥有了jQuery，同使用原生 JavaScript 编写跨浏览器代码相比，jQuery 是一次跳跃式的提升。你应该还记得，一个 AJAX 调用需要写多行代码，而且还必须考虑兼容 IE 浏览器和类 WebKit 浏览器。使用 jQuery，这只需要一个调用：例如$.ajax()。过去，jQuery 被称为一个框架，可如今它已经不再是了！现在，框架是一个更大且更强的概念。

与 Backbone 和 Angular 类似，每个新生代 JavaScript 框架都会带来很多新的优点。在这一点上 React 并非唯一。React 带来的革新是它挑战了大多数流行的前端框架所使用的一些核心概念：例如需要拥有模板这样的想法。

以下列表突出显示了 React 与其他库和框架相比的一些优点：

- 更简单的应用——React 使用纯 JavaScript 构建 CBA；声明式风格；强大且开发者友好的 DOM 抽象(不只是 DOM，还有 iOS、Android 等)。
- 快速的 UI——React 通过虚拟 DOM 和智能协调算法提供出色的性能，借助此优点，可以让你的性能测试不再需要启动一个无界面浏览器。
- 编写更少的代码——React 伟大的社区和丰富的组件生态为开发人员提供了各种库和组件。考虑框架选型时，这一点至关重要。

许多特点使得 React 相比其他大多数前端框架更容易使用。让我们逐个打开这些项目，从它的简单性开始。

1.3.1 简单性

计算机科学中的简单性概念受到开发人员和用户的高度认可。它不同于易用性。简单的事情可能实现起来会很困难，但最终会更加优雅和高效。通常，一件容易的事情最终会变得复杂。简单性与 KISS 原则(保持简单、傻瓜)[12]密切相关。要点是：越简单的系统工作得越好。

React 通过为软件工程师提供更好的 Web 开发体验，提供了一个更简单的解决方案。当我开始使用 React 时，经历了一次相当大的转变，我需要时刻提醒自己使用纯粹的、无框架的 JavaScript 而不是 jQuery。

在 React 中，这种简单性通过以下特性实现：

- **命令式风格之上的声明式风格**——React 通过自动更新视图来实现在命令式风格之上拥抱声明式风格。
- **使用纯 JavaScript 的基于组件的架构**——React 不会为其组件使用领域特定语言(Domain Specific Language，DSL)，只是纯 JavaScript。在实现同一功能时没有分离。
- **强大的抽象**——React 拥有与 DOM 进行交互的简化方法，允许进行规范化的事件处理和使用其他接口，就像在浏览器中一样。

让我们逐一介绍。

12 https://en.wikipedia.org/wiki/KISS_principle

命令式风格之上的声明式风格

首先，React 在命令式风格之上拥抱了声明式风格。声明式风格意味着开发人员编写代码以示事物应该是什么，而不是一步步(命令式)地去做什么。但为什么声明式风格是更好的选择呢？因为声明式风格降低了复杂性，使代码更容易阅读和理解。

考虑这个简短的 JavaScript 示例，其中说明了声明式和命令式编程之间的区别。假设需要创建一个数组(arr2)，它的元素是将另一个数组(arr)的元素加倍之后的结果。可以使用 for 循环遍历数组，并告诉系统将元素乘以 2 并创建一个新元素(arr2[i]=)：

```
var arr = [1, 2, 3, 4, 5],
  arr2 = []
for (var i=0; i<arr.length; i++) {
  arr2[i] = arr[i]*2
}
console.log('a', arr2)
```

这段代码的运行结果是将每个元素乘以 2 并按如下所示将结果输出到控制台上：

```
a [2, 4, 6, 8, 10]
```

这个例子阐述了命令式编程，它可以正常运行。但由于代码的复杂性，它有可能会变得无法正常运行。当有太多的命令式语句时，会变得难以理解最终的结果应该是什么。幸运的是，可以通过 map( )使用声明式写法重写这段逻辑：

```
var arr = [1, 2, 3, 4, 5],
  arr2 = arr.map(function(v, i){ return v*2 })
console.log('b', arr2)
```

输出为 b[2,4,6,8,10]。变量 arr2 与前面的例子相同。哪段代码更容易阅读和理解？我个人陋见是：采用声明式写法的那段。

以下命令式代码从一个对象中获取嵌套的值。表达式需要返回一个基于字符串(如 account 或 account.number)的值，使得这些语句输出 true：

```
var profile = {account: '47574416'}
var profileDeep = {account: { number: 47574416 }}
console.log(getNestedValueImperatively(profile, 'account') === '47574416')
console.log(getNestedValueImperatively(profileDeep, 'account.number')
➥ === 47574416)
```

命令式风格的代码从字面上告诉系统要做什么来获取你需要的结果：

```
var getNestedValueImperatively = function getNestedValueImperatively
➥ (object, propertyName) {
  var currentObject = object
  var propertyNamesList = propertyName.split('.')
  var maxNestedLevel = propertyNamesList.length
  var currentNestedLevel

  for (currentNestedLevel = 0; currentNestedLevel < maxNestedLevel;
➥ currentNestedLevel++) {
```

```
  if (!currentObject || typeof currentObject === 'undefined')
  ➥ return undefined
  currentObject = currentObject[propertyNamesList[currentNestedLevel]]
  }
  return currentObject
}
```

与声明式风格相比(关注结果)，局部变量的数量减少了，从而简化了逻辑：

```
var getValue = function getValue(object, propertyName) {
  return typeof object === 'undefined' ? undefined : object[propertyName]
}

var getNestedValueDeclaratively = function getNestedValueDeclaratively(object,
➥ propertyName) {
  return propertyName.split('.').reduce(getValue, object)
}
console.log(getNestedValueDeclaratively({bar: 'baz'}, 'bar') === 'baz')
console.log(getNestedValueDeclaratively({bar: { baz: 1 }}, 'bar.baz')=== 1)
```

大多数程序员都受过命令式风格的编码训练，但通常声明式代码更加简单。在这个例子中，使用较少的变量和语句可以使声明式代码更容易掌握。

那只是一些 JavaScript 代码。和 React 有什么关系呢？当组合 UI 时，React 采用相同的声明式风格。首先，React 开发人员以声明式风格描述 UI 元素。然后，当对这些 UI 元素生成的视图进行修改时，React 会自己完成更新。这太棒了！

当需要让视图发生变更时，React 声明式风格的便利性将被完全展现出来。这些变更被称为内部状态的变更。当变更发生时，React 会自动更新视图。

注意：第 4 章会介绍状态的工作原理。

在底层，React 使用虚拟 DOM 来查找浏览器中已有视图和新视图之间的差异(增量)。这个过程称为 DOM diff，又称为状态和视图的协调(使它们保持相似性)。这意味着开发人员不需要担心显式地修改视图，他们需要做的就是更新状态，视图会根据需要自动更新。

相反，使用 jQuery 则需要强制执行更新。通过操作 DOM，开发人员可以通过编程修改网页或网页的一部分(更可能的情况)，而无须重新渲染页面。调用 jQuery 方法时所做的就是进行 DOM 操作。

一些框架，比如 Angular，可以执行自动的视图更新。在 Angular 中，这被称为双向数据绑定，这基本上意味着视图和模型之间有双向的数据通信/同步。

jQuery 和 Angular 方式不是很好，有两个原因。可以把它们想象成两个极端：其中一个极端，jQuery 没有做任何事情，开发人员需要手动实现所有的更新；另一个极端，框架(Angular)正在做一切事情。

jQuery 方式很容易出错，需要做更多的工作来实现。此外，直接操作常规 DOM 的方式可以让简单的 UI 正常工作，但是当处理 DOM 树中的大量元素时，这是有限制的。情况就是这样，因为命令式函数比声明式语句更难以看到结果。

Angular 方式很难理解，因为双向绑定，事情可能会迅速失控。当加入越来越多的逻

辑时，可能突然间，不同的视图在更新模型，这些模型又更新了其他视图。

Angular 方式固然比命令式 jQuery 更加可读，而且需要手动编码的量也更少！但仍存在另一个问题。Angular 依赖使用 ng 指令(例如 ng-if)的模板和 DSL。下一节将讨论 Angular 方式的缺点。

使用纯 JavaScript 的基于组件的架构

基于组件的架构[13]在 React 之前就已存在。分离关注点、松散耦合和代码重用是这种架构的核心，因为它提供了许多好处。软件工程师(包括 Web 开发者)很喜欢 CBA。React 中的 CBA 构建单元是 Component 类。与其他 CBA 架构一样，它有许多好处，代码重用是主要的(可以编写更少的代码)。

在 React 之前，一直缺少此类架构的纯 JavaScript 实现。当使用 Angular、Backbone、Ember 或其他类 MVC 前端框架时，会有一个文件用于 JavaScript，另一个文件用于模板(Angular 对组件使用“指令”这个术语)。对于单个组件，存在两种语言(和两个或更多个文件)会有一些问题。

当需要在服务器端渲染 HTML 时，HTML 和 JavaScript 分离可以工作得很好，JavaScript 的作用仅仅是让文本闪烁起来。现在单页面应用(SPA)需要处理复杂的用户输入并在浏览器上执行渲染。这意味着 HTML 和 JavaScript 在功能上是紧密耦合的。对于开发人员而言，如果开发一个项目(组件)时，不再需要将 HTML 和 JavaScript 分离，这会更有意义。

来看下面这段 Angular 代码，它根据 user.session 的值来显示不同的链接：

```
<a ng-if="user.session" href="/logout">Logout</a>
<a ng-if="!user.session" href="/login">Login</a>
```

虽然可以读懂，但可能会疑惑 ng-if 需要什么类型的值：布尔值抑或字符串。会隐藏元素还是仅仅不渲染？在 Angular 案例中，无法确定元素在值为 true 或 false 时是否将被隐藏，除非熟悉这个特定 ng-if 指令的工作原理。

将上述代码与下面使用 if/else 实现条件渲染的 React 代码进行比较。user.session 的值应该是什么，并且如果 user.session 的值为 true，哪个元素会被渲染(logout 或 login)，对此毫无疑问。为什么？因为这只是 JavaScript：

```
if (user.session) return React.createElement('a', {href: '/logout'}, 'Logout')
else return React.createElement('a', {href: '/login'}, 'Login')
```

当需要遍历数组并打印属性时，模板很有用。我们一直在和数据列表打交道，下面来看看 Angular 中的 for 循环。如前所述，在 Angular 中，需要使用带有指令的 DSL，for 循环的指令是 ng-repeat：

```
<div ng-repeat="account in accounts">
  {{account.name}}
</div>
```

模板存在的一个问题是开发人员经常需要学习另外一种语言。在 React 中，使用的是纯 JavaScript，这意味着不需要学习新语言！下面是使用纯 JavaScript 编写账户名称列表的

13　http://mng.bz/a65r

UI 示例：

```
accounts.map(function(account) {
  return React.createElement('div', null, account.name)
})
```

将迭代器表达式作为参数的常规 JavaScript 方法[14]

迭代器表达式返回一个使用 标签包裹的账户名称

想象一下，假设正在对账户列表进行一些修改。需要显示账户数量和其他字段。除了 name 之外，如何才能知道账户中还有哪些其他字段呢？

需要打开调用和使用该模板的 JavaScript 文件，然后找到 accounts 对象以查看其属性。因此，模板存在的第二个问题是：数据的逻辑和如何呈现数据的描述是分开的。

将 JavaScript 和标记放在一起会更好，因为不必在文件和语言之间切换，这正是 React 的工作原理。你将在 Hello World 示例中看到 React 如何快速渲染元素。

注意：分离关注点通常会是一种很好的模式。简言之，它意味着分离不同的功能，如数据服务、视图层等。当使用模板标记和对应的 JavaScript 代码时，是在处理同一个功能问题，这就是为什么虽有两个文件(.js 和.html)却不是关注点分离的原因。

现在，如果要在渲染列表中显式地设置跟踪条目的方法(例如，确保列表中没有重复项)，可以使用 Angular 的 track by 特性：

```
<div ng-repeat="account in accounts track by account._id">
  {{account.name}}
</div>
```

如果想跟踪数组的下标，可以使用$index：

```
<div ng-repeat="account in accounts track by $index">
  {{account.name}}
</div>
```

但是，我和其他许多开发者关心的是，$index 背后的魔法到底是什么？在 React 中，可以使用 map( )的参数作为 key 属性的值：

```
accounts.map(function(account, index) {
  return React.createElement('div', {key: index}, account.name)
})
```

使用 Array.map( )提供的数组元素值(account)

返回一个 React 元素 ，里面包含一个值为 index 的 key 属性，并且设置内部文本为 account.name

值得注意的是：map( )不是 React 独有的，可以将它与其他框架结合使用，因为它是 JavaScript 语言的一部分。但是 map( )天生的声明式特点使它和 React 成为完美组合。

我没有选择 Angular，虽然它是一个很好的框架。如果一个框架使用 DSL，那么势必需要学习它的魔法变量和方法，这超出了我的底线。而在 React 中，可以使用纯 JavaScript。

如果选择 React，那么即使不在 React 中，也可以将所学知识应用到下一个项目中。另一方面，如果使用 X 模板引擎(或 Y 框架内置的 DSL 模板引擎)，你将被锁定在该系统中，

14 http://mng.bz/555J

并将自己描述成 X/Y 开发人员。你所学的知识无法在不使用 X/Y 的项目中应用。总而言之，基于纯 JavaScript 组件的架构关注的是使用独立、封装良好的可重用组件，从而可以确保实现更好的关注点分离，而不需要 DSL、模板或指令。

在与许多开发团队合作后，我观察到与简单性相关的另一个因素。与 MVC 框架(React 不是 MVC 框架，所以我会停止这种比较)和具有特殊语法的模板引擎(例如 Angular 指令和 Jade/Pug)相比，React 有一条更好、更浅、更渐进式的学习曲线。原因在于，大多数模板引擎使用自己的 DSL 来建立抽象，使用类似 if 条件和 for 循环的方式而不是使用 JavaScript 重塑这些事情。

强大的抽象

React 拥有强大的文档模型抽象功能，换言之，它隐藏了底层接口并提供归一化/合成的方法和属性。例如，当在 React 中创建 onClick 事件时，事件处理程序将不会收到浏览器原生的特定事件对象，而是收到封装了原生事件对象的合成事件对象。无论在什么浏览器中运行代码，都可以期待合成事件具有相同的行为。React 还有一组触摸事件的合成事件，这对于构建移动设备的 Web 应用非常有用。

React DOM 抽象的另一个例子是可以在服务器端渲染 React 元素，这可以方便搜索引擎优化(SEO)和/或提高性能。

相比渲染服务器后端返回的 HTML 字符串和 DOM，使用 React 组件进行渲染时有更多的选择。我们将在 1.5.1 节中介绍。并且关于 DOM，React 最受追捧的优点之一就是其卓越的性能表现。

1.3.2　速度和可测试性

除了必要的 DOM 更新，你的框架可能会执行一些不必要的更新，这使得复杂 UI 的性能变得更糟。当网页上有很多动态 UI 元素时，用户会受较大影响且很痛苦。

另一方面，React 的虚拟 DOM 只存在于 JavaScript 内存中。每次有数据更改时，React 首先使用虚拟 DOM 比较差异，只有当 React 知道渲染将会有更改时才会更新实际的 DOM。图 1.1 展示了有数据更改时对 React 虚拟 DOM 工作原理的高级概述。

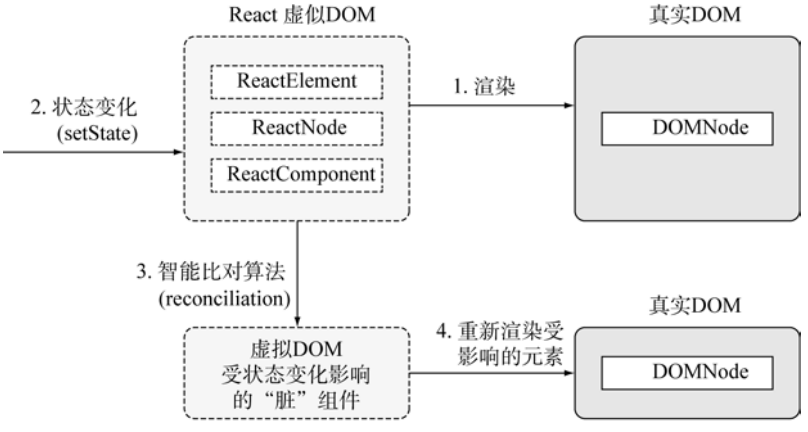

图 1.1　一旦组件被渲染，如果它的状态发生变化，就与内存中的虚拟 DOM 进行比较，在需要的时候重新进行渲染

最终，React 仅更新绝对必要的部分，以使内部状态(虚拟 DOM)和视图(真实 DOM)保持一致。例如，一个<p>元素，使用组件状态为其添加文本，则只会更新文本(即 innerHTML)，而不是更新元素本身。与渲染整个元素集合(甚至整个页面(服务器端渲染))相比，这样做会提高性能。

注意：如果喜欢挑战算法和算法复杂度，有两篇好文解释了 React 团队如何设法将 O(n3)问题变成 O(n)问题: “Reconcliliation”，参考 React 官网(http://mng.bz/PQ9X); 以及 Christopher Chedeau 所写的 “React’s Diff Algorithm”，参考 http://mng.bz/68L4。

虚拟 DOM 的另一个好处就是可以脱离诸如 PhantomJS(http://phantomjs.org)这样的无界面浏览器进行单元测试。有一个名为 Jest(https://facebook.github.io/jest)的 Jasmine (http://jasmine.github.io)层，可以让你在命令行上测试 React 组件。

1.3.3 生态和社区

最后，但并非最不重要的一点是：React 受 Facebook 开发人员以及 Instagram 的同行们支持，与 Angular 和其他库一样，背后拥有一家大公司提供健全的测试基础(已被部署到数百万浏览器中)、对未来提供保证以及对贡献速度的提升。

React 社区让人难以置信，大多数情况下，开发人员甚至不必自己实现大量的代码。看看这些社区资源：

- React 组件清单：https://github.com/brillout/awesome-react-components 和 http://devarchy.com/react-components
- 一套实现 Google Material Design 规范(https://design.google.com)的 React 组件库：http://react-toolbox.com
- Material Design React 组件库：www.material-ui.com
- 遵循 Office 设计语言以及 Office 和 Office 360 体验的 React 组件集合：https://github.com/OfficeDev/office-ui-fabric-react
- 开源 JS(大部分是 React)软件包的意见目录：https://js.coach
- React 组件目录：https://react.rocks
- Khan 学院 React 组件库：https://khan.github.io/react-components
- 注册 React 组件：www.reactjsx.com

开源的个人轶事和经验告诉我，开源项目的营销与代码本身的广泛采用和成功一样重要。也就是说，如果一个项目的网站不好，缺少文档和例子，并且 logo 丑陋，那么大多数开发者不会认真对待它，特别是在有如此多 JavaScript 库的当下。开发人员非常挑剔，他们不会使用“丑小鸭”库。

我的老师曾经说过，“不要用封面来判定一本书”，这听起来有争议，但令人遗憾的是，包括软件工程师在内的大多数人都倾向于偏好品牌。幸运的是，React 拥有很好的工程师知名度。

1.4　React 的缺点

当然，一切事物都有缺点，React 也不例外，但完整的缺点列表取决于询问的对象。不同的人会有一些差异，比如声明式和命令式是非常主观的。因此，它们既可为利，亦可为弊。下面是我的 React 缺点清单(如同任何类似的清单，这可能有偏见，因为这是基于从其他开发人员那里听到的意见)：

- React 不是一个完备的、瑞士军刀型框架。开发人员需要将其与 Redux 或 React Router 等库配合使用，以实现与 Angular 或 Ember 相当的功能。如果需要一个简约的 UI 库来与现有技术栈集成，这也可以成为优势。
- React 不如其他框架那么成熟，它的核心 API 仍然在变化，尽管 0.14 版本之后变化很少。React 的最佳实践(以及组件、插件和附加组件的生态系统)仍然在持续发展。
- React 使用一种新的 Web 开发方法，JSX 和 Flux(通常用作 React 的数据管理类库)可能会对初学者有一定的门槛。缺乏可用于掌握 React 的最佳实践、好书、课程和资源。
- React 只有单向绑定。虽然单向绑定对于复杂应用更好，并且消除了大量的复杂性，但是一些开发人员(特别是 Angular 开发人员)已习惯双向绑定，他们会发现自己需要编写更多的代码。我将在第 14 章比较 React 的单向绑定和 Angular 的双向绑定，包括数据处理。
- React 不是开箱即用的响应式库(比如在响应式编程和架构中，它们更具备事件驱动、弹性和响应性的特点)。开发人员需要使用其他工具(例如 Reactive Extensions (RxJS，https://github.com/Reactive-Extensions/RxJS))来构建支持 Observables 的异步数据流。

继续上述对 React 的介绍，我们来看看如何将之集成到一个 Web 应用中。

1.5　React 如何与 Web 应用集成

某种程度上，React 本身没有 React Router 或数据处理库，和框架(如 Backbone、Ember 和 Angular)的可比性更小，更适合和处理 UI 的库进行比较，例如模板引擎(Handlebars、Blaze)和 DOM 操作类库(jQuery、Zepto)。实际上，许多团队已经将传统的模板引擎(如 Backbone 的 Underscore 或 Meteor 的 Blaze)切换到 React，并取得非常大的成功。例如，Paypal 从 Dust 转为 React，与本章前面列出的许多其他公司一样。

可以针对部分 UI 使用 React。例如，假设有一个使用 jQuery 构建的贷款申请表页面。可以逐渐地将这个前端应用转换为采用 React，首先把城市和州名字段转换为根据邮政编码自动填充。表单其余部分可以继续使用 jQuery。然后，如果要继续，可将其余表单元素从 jQuery 转换为 React，直到整个页面都基于 React。采用类似的方法，许多团队将 React 与 Backbone、Angular 或其他现有的前端框架成功地整合在了一起。

对前端开发而言，React 不关注后端技术选型。换言之，无须依赖 Node.js 后端或 MERN(MongoDB、Express、React 和 Node.js)来使用 React。React 和其他后端技术(如

Java、Ruby、Go、Python)配合得同样很好。毕竟 React 只是一个 UI 库。可以将其与任何后端和前端数据类库(Backbone、Angular、Meteor)集成在一起。

总结一下 React 如何与 Web 应用集成，最常见于下列情况：

- 作为 React 单页面技术栈中的 UI 库，例如 React+React 和 Router+Redux。
- 作为非完整 React 单页面技术栈中的 UI 库(MVC 中的 V)，例如 React+Backbone。
- 作为任何前端技术栈中的 UI 组件，例如 jQuery+服务器端渲染技术栈中的 React 自动完成输入组件。
- 作为传统的富服务器端 Web 应用、混合 Web 应用或同构 Web 应用的服务器端模板库。
- 作为移动应用中的 UI 库，例如 React Native iOS 应用。
- 作为不同渲染目标的 UI 描述库(在下一节中讨论)。

React 与其他前端技术一起配合得很好，但它主要用作单页面架构(Single-Page Architecture，SPA)的一部分，因为 SPA 似乎是构建 Web 应用最有利和最受欢迎的方案。我会在 1.5.2 节中介绍如何将 React 集成到 SPA 中。

在某些极端情况下，甚至可以在服务器端使用 React 作为模板引擎。例如，有一个名为 express-react-views 的库(https://github.com/reactjs/express-react-views)，它在服务器端从 React 组件中渲染视图。这种服务器端渲染是可行的，因为 React 允许使用不同的渲染目标。

1.5.1 React 类库和渲染目标

在版本 0.14 和更高的版本中，React 团队将库拆分成两个包：React Core(npm 上的 react 包)和 ReactDom(npm 上的 react-dom 包)。React 维护者通过这种方式清楚地表明：React 不仅是一个 Web 库，而且是正在变成一个描绘 UI 的通用库(有时称之为同构的，因为它可以在不同环境中使用)。

例如，在版本 0.13 中，React 使用 React.render( )方法将元素挂载到网页的 DOM 节点上。在版本 0.14 及更高的版本中，需要依赖 react-dom 包，并调用 ReactDOM.render( )方法而不是 React.render( )。

React 社区创建了许多包来支持各种渲染目标，这些包的存在使得这种分离组件书写和组件渲染逻辑的方法成为可能。其中一些模块如下：

- blessed(https://github.com/chjj/blessed)终端控制台接口渲染器：http://github.com/Yomguithereal/react-blessed
- 用于 ART 库的渲染器(https://github.com/sebmarkbage/art)：https://github.com/reactjs/react-art
- 用于<canvas>的渲染器：https://github.com/Flipboard/react-canvas
- 使用了 three.js 的 3D 库渲染器(http://threejs.org)：https://github.com/Izzimach/react-three
- 虚拟现实和 360°互动体验渲染器：https://facebook.github.io/react-vr

除了这些库的支持外，React Core 和 ReactDOM 的分离使得在 React 和 React Native 库(用于原生移动 iOS 和 Android 开发)之间共享代码变得更加容易。实际上，使用 React 进行 Web 开发时，需要至少包含 React Core 和 ReactDOM。

此外，React 和 npm 中还有 React 的其他实用程序库。在 React v15.5 版本之前，其中

一些被作为 React 附加组件[15]的一部分。这些实用程序库允许对功能进行增强、使用不可变数据(https://github.com/kolodny/immutability-helper)和进行测试。

最后，React 几乎总是和 JSX 一起使用，这是一种小巧的语言，可以让开发人员更加高效地编写 React UI。可以使用 Babel 或类似的工具将 JSX 转换为常规的 JavaScript。

如你所见，和 React 相关的功能被分成很多不同的模块和 npm 包。这是一件好事，让你有更多的能力和选择，而不是通过准备一个完整的、大一统的库来为满足需求提供唯一可能的方式。更多内容会在 1.5.3 节中介绍。

如果你是一名 Web 开发者，可能会使用 SPA 架构。也许你已经有一个 SPA 架构的 Web 应用，并且希望使用 React 重新改造它；也许你准备从头开始一个新项目。接下来，我们将放大 React 在 SPA 中的地位，把它作为最流行的 Web 应用构建方法。

1.5.2　单页面应用和 React

SPA 架构的别名是富客户端，因为作为客户端的浏览器拥有更多的逻辑，比如渲染 HTML、校验、UI 变更等功能。图 1.2 是一幅包含用户、浏览器和服务器的典型 SPA 架构图。图 1.2 中描绘了一个用户正在发出一个请求，以及诸如单击按钮、拖曳、鼠标悬停等输入行为。

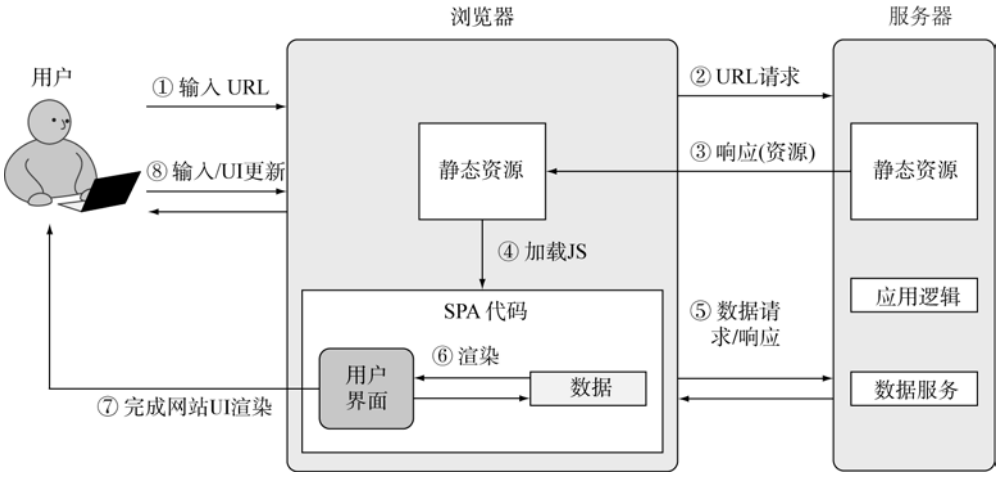

图 1.2　典型的 SPA 架构

① 用户在浏览器中输入一个 URL 以打开一个新的页面。

② 浏览器向服务器发送 URL 请求。

③ 服务器使用静态资源(如 HTML、CSS 和 JavaScript)进行响应。在大多数情况下，HTML 只是概要，里面只有网页的骨架。通常会显示消息“加载中……”和/或转动的 GIF 图片。

④ 静态资源包括 SPA 中的 JavaScript 代码。加载时，这些 JavaScript 代码对数据进行附加的请求(AJAX/XHR 请求)。

15　查看版本15.5的变更日志，其中包含附加组件和npm库的清单：https://facebook.github.io/react/blog/2017/04/07/react-v15.5.0.html。也可以查看附加组件页面：https://facebook.github.io/react/docs/addons.html。

⑤ 数据以 JSON、XML 或任何其他格式返回。

⑥ 一旦 SPA 接收到数据，就可以渲染缺失的 HTML 结构(图 1.2 中的用户界面模块)。换言之，UI 渲染发生在浏览器端，并通过向 SPA 模板填充数据来完成[16]。

⑦ 一旦浏览器完成渲染，SPA 将替换消息“加载中……”用户便可以使用该页面了。

⑧ 用户看到一个漂亮的网页，并可以与页面(图 1.2 中的输入)交互，触发从 SPA 到服务器的新请求，继续步骤②～⑥的循环过程。在这个阶段，如果 SPA 实现了路由功能，那么可能会触发路由的变更，这意味着导航到一个新的 URL 会在浏览器中触发一次 SPA 渲染，而不是从服务器端重新加载一个新的页面。

总而言之，在 SPA 方式中，UI 的大部分渲染发生在浏览器端。只有数据往返于浏览器。相比而言，使用富服务器端方式，所有渲染都发生在服务器端(这里的渲染，指的是从模板或 UI 代码生成 HTML，而不是在浏览器中渲染 HTML(有时也称为绘制 DOM))。

请注意，类 MVC 架构是实现 SPA 最流行的方式，但并不是唯一的方式。React 不需要使用类 MVC 架构，但为了简单起见，我们假设你的 SPA 正在使用类 MVC 架构。可以在图 1.3 中看到其可能存在的不同部分。导航器或路由库作为 MVC 范例中的控制器，指定要获取的数据和要使用的模板。导航器/控制器发起获取数据的请求，然后使用数据对模板(视图)进行填充，以 HTML 的形式渲染 UI。UI 会将操作(单击、鼠标悬停、敲击键盘等)发送回 SPA 代码。

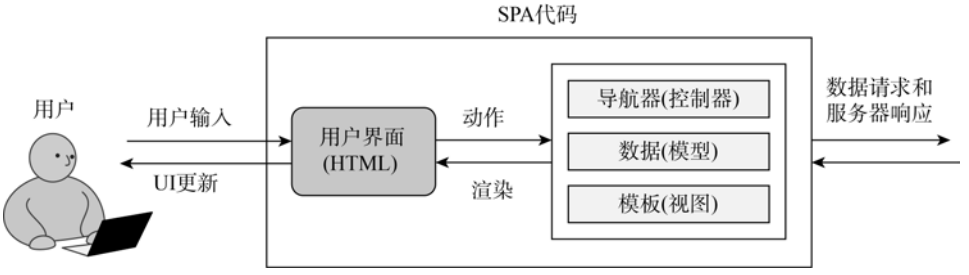

图 1.3 单页面应用的内部工作情况

在 SPA 架构中，数据在浏览器中进行解析和处理(浏览器渲染)，并被用于渲染额外的 HTML 或更改已有的 HTML。这使得 SPA 成为交互体验可以与桌面应用相媲美的 Web 应用。Angular、Backbone 和 Ember 是适合构建 SPA 的前端框架范例。

注意：不同的框架实现了不同的导航器、数据和模板，因此图 1.3 并不适用于所有框架。相反，图 1.3 只是阐述了典型 SPA 架构中最广泛的关注点分离。

React 在图 1.3 所示的 SPA 图表中所处的位置是模板方块。React 是一个视图层，因此可以通过向其提供数据来使它渲染 HTML。当然，React 相比典型的模板引擎做的要多很多。React 和其他模板引擎(如 Underscore、Handlebars 和 Mustache)之间的区别在于开发、更新 UI 和管理状态的方式。我们将在第 4 章中更详细地讨论状态。现在，将状态视为和 UI 相关的可更改数据即可。

16 “What does it mean to hydrate an object?”，Stack Overflow, http://mng.bz/uP25

1.5.3　React 技术栈

React 并不是一个完整的前端 JavaScript 框架。它非常简约，并不强制使用特定方式实现诸如数据模型、样式和路由等功能。因此，开发人员需要将 React 与路由和/或模型库搭配使用。

例如，已经使用 Backbone 和 Underscore 模板引擎的项目可以切换到 Underscore for React，并保留现有的数据模型和 Backbone 路由(Underscore 还包含了实用程序，而不仅仅包含模板方法，可以将这些 Underscore 实用程序与 React 一起用作声明式风格的解决方案)。其他时候，开发人员选择使用 React 技术栈，这个技术栈由专门为 React 创建的数据和路由库组成。

- 数据模型类库和后端：RefluxJS(https://github.com/reflux/refluxjs)、Redux(http://redux.js.org)、Meteor(https://www.meteor.com)和 Flux(https://github.com/facebook/flux)
- 路由库：React Router(https://github.com/ReactTraining/react-router)
- 使用 Twitter Bootstrap 库的 React 组件集合：React-Bootstrap(https://react-bootstrap.github.io)

React 组件库的生态系统每天都在增长。React 描述可组合组件的能力非常有助于代码重用。有许多组件被打包成 npm 模块，只是为了证明拆分成小的可组合组件对于代码重用有好处，这里有一些受欢迎的 React 组件：

- 日期选择组件：https://github.com/Hacker0x01/react-datepicker
- 一套处理表单呈现和校验的工具：https://github.com/prometheusresearch/react-forms
- WAI-ARIA 兼容的自动完成组件：https://github.com/reactjs/react-autocomplete

接下来，我们聊聊 JSX，JSX 可能是最常见的不愿使用 React 的理由。如果熟悉 Angular，那么可能已经在模板代码中编写了大量的 JavaScript 代码。这是因为在现代 Web 开发中，纯 HTML 过于静态并且难以使用。我的建议是：对 React 的优点存疑，给 JSX 一次公平的展现机会。

JSX 使用和 XML/HTML 一样的语法，通过<>在 JavaScript 中编写 React 对象。React 和 JSX 配合得很好，因为开发人员可以更好地实现和阅读代码。JSX 可以看成编译成原生 JavaScript 的迷你语言，因此，JSX 并不在浏览器中运行，而是作为待编译的源代码。下面是用 JSX 编写的一段紧凑的代码：

```
if (user.session)
  return <a href="/logout">Logout</a>
else
  return <a href="/login">Login</a>
```

即便在浏览器中加载 JSX 文件，并通过运行时转换库在运行时将 JSX 编译为原生 JavaScript，也仍然不是运行 JSX，而是运行 JavaScript。在这一点上，JSX 类似于 CoffeeScript。将这些语言编译为原生 JavaScript 以获得相比 JavaScript 提供的更好的语法和功能。

对某些人而言，将 XML 插入到 JavaScript 代码中看起来非常怪异。我花了一段时间进行调整，因为我期待着一场雪崩式的语法错误消息。另外，JSX 是可选的。鉴于这两个原因，在第 3 章之前不会介绍 JSX；不过相信我，一旦掌握，功能会非常强大。

迄今为止，你已经了解了 React 是什么，它的技术栈以及它在更高层次 SPA 架构中的位置。现在是时候卷起袖子，开始编写你的第一个 React 项目了。

1.6 第一个 React 项目：Hello World

让我们开始探讨你的第一个 React 项目——学习编程语言的典型示例——HelloWorld 程序。这里不会使用 JSX，有的只是简单的 JavaScript。这个项目会在网页上打印内容为“Hello world!!!”的<h1>标题。图 1.4 展示了完成后的效果。

图 1.4 Hello World

首先学习不包含 JSX 的 React

尽管大多数 React 开发者都使用 JSX，但浏览器只能运行标准的 JavaScript。这也是理解纯 JavaScript 中的 React 代码为什么有益的原因。之所以从简单的 JS 开始，另一个原因是为了展示 JSX 是可选的，尽管它是 React 事实上的标准语言。最后一个原因是：JSX 的预处理需要一些工具。

我想尽快让你开始使用 React，而不用在本章中花费太多时间进行设置。在第 3 章中会执行 JSX 所有必要的设置。

该项目的文件夹结构很简单，它由 js 文件夹中的两个 JavaScript 文件和一个 HTML 文件 index.html 组成：

```
/hello-world
  /js
    react.js
    react-dom.js
index.html
```

js 文件夹中的两个文件是 React v15.5.4[17]版本的：react-dom.js(Web 浏览器的 DOM 渲染器)和 react.js(React Core 包)。首先，需要下载上述 React Core 和 ReactDOM 库。方法有很多种，建议使用本书源代码中提供的文件，请访问 www.manning.com/books/react-quickly 和 https://github.com/azat-co/react-quickly/tree/master/ch01/hello-world。这是最可靠且

17 v15.5.4是本书写作时的最新版本。通常，像14、15和16这样的主要版本间包含显着差异，而像15.5.3和15.5.4这样的小版本间变化和冲突较少。本书的代码在版本v15.5.4下进行了测试。这些代码可与未来的版本一起运行，但不能保证它们能正常运行。因为没有人知道未来的版本会是什么样的，甚至核心贡献者也不知道。

简单的方法，因为不需要依赖任何其他服务和工具。可以在附录 A 中找到下载 React 的更多方式。

警告：在版本 0.14 之前，这两个库是捆绑在一起的。例如，对于版本 0.13.3，只需要引入 react.js 即可。除非另有说明，本书使用版本 15.5.4(本书撰写时的最新版本)的 React 和 ReactDOM。对于本书第 I 部分的大多数项目，将需要两个文件：react.js 和 react-dom.js。在第 8 章中，将需要 prop-types(www.npmjs.com/package/prop-types)包，在版本 15.5.4 以前该包是 React 的一部分，但现在是一个单独的模块。

将 React 文件放在 js 文件夹中之后，在 Hello World 项目文件夹中创建 index.html 文件。这个 HTML 文件将是 Hello World 应用的入口(意味着需要在浏览器中打开)。

index.html 的代码很简单，首先在<head>标签中包含 React 相关的库。在<body>元素中，创建一个 id 属性为 content 的 容器，以及一个<script>标签(该应用的代码会写在这里)，如下所示：

代码清单1.1　加载React库和代码(index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <script src="js/react.js"></script>
    <script src="js/react-dom.js"></script>
  </head>
  <body>
    <div id="content"></div>
    <script type="text/javascript">
        ...
    </script>
  </body>
</html>
```

导入React库

导入ReactDOM库

定义一个空的 元素以便挂载React UI

开始编写Hello World视图的React代码

为什么不直接在<body>元素中渲染 React 元素呢？因为这样做可能会导致与操作 document body 的其他库和浏览器扩展程序发生冲突。实际上，如果尝试把 React 元素挂载到 body 上，会得到如下警告：

```
Rendering components directly into document.body is discouraged...
```

这是 React 的另一个优点：提供很好的警告和错误消息！

注意：React 的警告和错误信息不包含在生产构建版本中，目的是减少干扰、增加安全性并尽可能减少发布大小。生产版本是 React Core 库的最小化版本，例如 react.min.js。有警告和错误消息的开发版本是未经最小化的版本，例如 react.js。

通过在 HTML 中引入如下两个库，可以访问 React 和 ReactDOM 这两个全局对象：window.React 和 window.ReactDOM。我们还需要两个方法：一个用于创建 React 元素(React)，另一个用于在 容器(ReactDOM)中渲染 React 元素，如代码清单 1.2 所示。

要创建 React 元素，需要做的就是调用 React.createElement(elementName, data, child)，三个参数的含义如下：

- elementName：HTML 标签字符串(例如'h1')或自定义的组件类对象(例如 2.2 节中的 HelloWorld)。
- data：特性和属性数据(稍后会介绍)，例如 null 或{name: 'Azat'}。
- child：子元素或内部 HTML/文本内容，例如“Hello world!”。

代码清单1.2 创建和渲染h1元素(index.html)

```
var h1 = React.createElement('h1', null, 'Hello world!')
ReactDOM.render(
  h1,
  document.getElementById('content')
)
```

创建和保存一个存储在变量h1中的React元素

在ID为content的真实DOM元素中渲染h1元素

上面代码清单获取 h1 类型的 React 元素，并将该对象引用存储到 h1 变量中。h1 变量不是实际的 DOM 节点，而是 React 的 h1 组件(元素)的实例。可以以任意方式命名它，例如 helloWordHeading。换言之，React 提供对 DOM 的抽象。

注意：h1 变量名是随意命名的。可以将其命名为任何想要的内容(例如 bananza)，只需要在 ReactDOM.render( )方法中使用相同的变量。

一旦元素被创建并存储在 h1 变量中，就可以使用代码清单 1.2 所示的 ReactDOM.render( )方法，将其渲染到 id 为 content 的 DOM 节点上。如果愿意，也可以把 h1 变量的表达式移到 render 调用中，除了不使用额外的变量之外，结果是相同的：

```
ReactDOM.render(
  React.createElement('h1', null, 'Hello world!'),
  document.getElementById('content')
)
```

现在，在你最喜欢的浏览器中打开由静态 HTTP 服务器发布的 index.html 文件。建议使用最新版本的 Chrome、Safari 或 Firefox 浏览器。你应该会在网页上看到“Hello world!”，如图 1.5 所示。

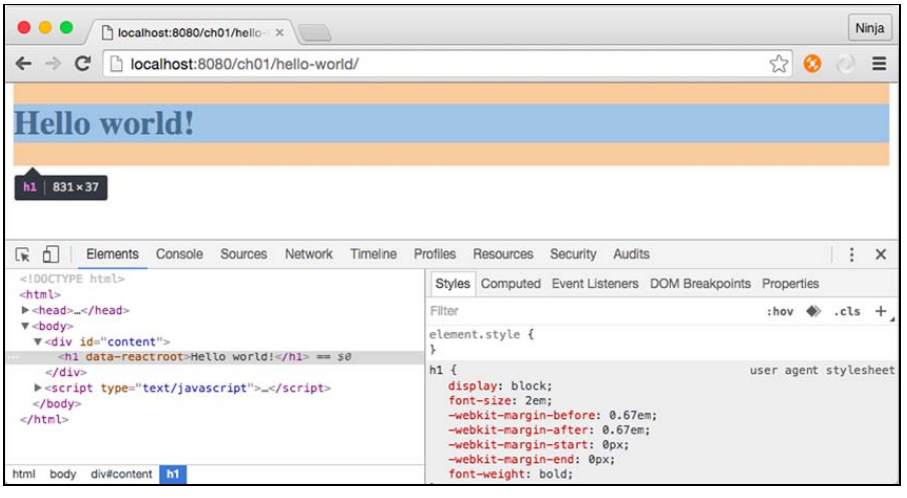

图 1.5 检查由 React 渲染的 Hello World 应用

图 1.5 展示了 Chrome 开发者工具的 Elements 选项卡并选择了<h1>元素。可以观察 data-reactroot 属性，它表示该元素由 ReactDOM 渲染。

需要注意的是：可以将 React 代码(代码清单 1.2)抽象为一个独立的文件，而不是创建一个元素，并在同一个 index.html 文件(代码清单 1.1)中使用 ReactDOM.render( )渲染它们。例如，可以创建 script.js 并将 h1 元素和 ReactDOM.render( )调用复制并粘贴到该文件中。然后在 index.html 中，需要在 id 为 content 的 标签后面插入 script.js，如下所示：

```
<div id="content"></div>
<script src="script.js"></script>
```

本地开发 Web 服务器

最好通过本地 Web 服务器访问 index.html 文件，而不是直接在浏览器中打开。因为使用 Web 服务器，JavaScript 应用可以生成 AJAX/XHR 请求。可以通过查看地址栏中的 URL 来判断是通过 Web 服务器访问还是直接访问文件。如果地址以 file 开头，就是直接访问文件；如果以 http 开头，就是通过 Web 服务器访问。在后续的项目中需要用到这个特性。通常，本地 HTTP Web 服务器会监听对 127.0.0.1 或 localhost 的请求。

可以使用任意的开源 Web 服务器，例如 Apache、MAMP.(我的最爱，因为它们是使用 Node.js 开发的)node-static(https://github.com/cloud-head/node-static)或 http-server(www. npmjs.com/package/http-server)。为了安装 node-static 或 http-server，需要安装 Node.js 和 npm。如果还没有它们，可以在附录 A 中找到 Node 和 npm 的安装说明，或者访问 http://nodejs.org。

假设你的机器上安装有 Node.js 和 npm，在命令提示符下运行 npm i –g node-static 或 npm i –g http-server。然后，切换到项目的源代码目录，运行 static 或 http-server。这里是从 react-quickly 目录开始执行的，所以需要在浏览器的地址栏中追加 Hello World 的路径：http://localhost:8080/ch01/hello-world/(见前面的图 1.5)。

恭喜！你刚刚实现了第一个 React 项目！

1.7　测验

1. 声明式编程不允许存储的值发生改变，这就是“要什么”，而不是命令式编程的“怎么做”。这么理解正确还是错误？

2. 要把 React 组件渲染到 DOM 中，使用哪个方法？(注意，这是个棘手的问题) ReactDOM.renderComponent、React.render、ReactDOM.append 还是 ReactDOM.render？

3. 必须在服务器上使用 Node.js 才能在 SPA 中使用 React。这么理解正确还是错误？

4. 必须包含 react-dom.js 才能在网页上渲染 React 元素。这么理解正确还是错误？

5. React 解决的问题是在数据变更时更新视图。这么理解正确还是错误？

1.8　小结

- React 是声明式的，它只是一个视图或 UI 层。

- React 使用组件，并且通过 ReactDOM.render( )方法对其进行渲染。
- React 组件类通过 class 创建，只有 render( )方法是必要的。
- React 组件是可重用的，并且拥有不可变的属性，可以通过 this.props.NAME 来访问它们。
- 在 React 中使用纯 JavaScript 来编写和组合 UI。
- 进行 React 开发时不必非得使用 JSX(针对 React 对象的一种类 XML 语法)，JSX 是可选的。
- 总结 React 的定义：针对 Web 的 React 由 React Core 库和 ReactDOM 库组成。React Core 库旨在通过使用 JavaScript 和(可选的)JSX，以同构的方式构建和共享可组合的 UI 组件。另一方面，要在浏览器中使用 React，可以使用 ReactDOM 库，该库拥有用于 DOM 渲染和服务器端渲染的方法。

1.9 测验答案

1. 正确。声明式风格的特点是“这是我想要的”，而命令式风格的特点是“这是怎么做的”。
2. ReactDOM.render。
3. 错误。可以使用任何后端技术。
4. 正确。需要 ReactDOM 库。
5. 正确。这正是 React 要解决的主要问题。