Web 开发与设计

React Hooks 实战

[英] 约翰·拉森(John Larsen) 著

周 轶 张兆阳 颜 宇 译

清華大學出版社

北 京

React Hooks in Action，With Suspense and Concurrent Mode
EISBN: 978-161729-763-2

北京市版权局著作权合同登记号　图字：01-2022-4301

图书在版编目(CIP)数据

React Hooks实战 / (英) 约翰·拉森(John Larsen) 著；周轶，张兆阳，颜宇译. —北京：清华大学出版社，2022.8
(Web 开发与设计)
书名原文：React Hooks in Action，With Suspense and Concurrent Mode
ISBN 978-7-302-61357-2

I. ①R…　II. ①约…　②周…　③张…　④颜…　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(2022)第 124199 号

责任编辑：王　军
封面设计：孔祥峰
版式设计：思创景点
责任校对：成凤进
责任印制：

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：010-83470000　　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：
经　　销：全国新华书店
开　　本：170mm×240mm　　印　　张：21　　字　　数：412 千字
版　　次：2022 年 8 月第 1 版　　印　　次：2022 年 8 月第 1 次印刷
定　　价：98.00 元

产品编号：094311-01

译 者 序

React 是一个专注于渲染的 UI 引擎，自 2013 年开源以来，受到众多开发者的青睐。它采用了数据驱动的思想，开发者可以将更多的注意力放在数据和状态的变化而非 DOM 操作上，同时它是组件化的，这使得它成为构建快速响应的大型 Web 应用程序的首选方式。随着 React 技术的发展和社区的壮大，开发者们也发现了一些问题。React 是以 UI 组件形式构建应用程序的，虽然组件可以复用，但其内部带有状态的逻辑并不能提取到函数或被其他组件复用，这导致项目中经常会出现巨大的单体组件，并且在不同的组件中可能留有相同且重复的代码逻辑。为此 React 也提出了一些特有的模式以解决此类问题，如 render prop 和高阶组件(higher-order component)，但是这些模式的引入提高了复杂性且不易理解。于是 React Hooks 应运而生，它使我们能够以可复用且独立的单元来组织组件内部的逻辑，我们称之为“逻辑关注点分离”。它不会像 render prop 和高阶组件那样引入不必要的、嵌套的组件树，我们也不必再承受 mixins 之苦。它是简单且独立的。

本书是一本介绍前端 React Hooks 的书籍，目前市面上介绍 React 的书不少，但是全面、完整地讲解 React Hooks 的中文书籍不多。作为 React 后续更新的重大特性，正确地使用 React Hooks 是每一位 React 开发者必备的技能。本书通过一个可运行的预订应用程序为示例，详细介绍了各种 hook 的使用场景和方法——这些 hook 包括 useState、useReducer、useEffect、useRef、useCallback、useMemo、useContext、自定义 hook 以及常用的第三方 hook，同时在进阶内容中介绍了 Suspense 组件和 Concurrent 模式下的实验性 API。本书为书中展示的代码示例都提供了相应的链接，并对关键代码添加了注释，以帮助你理解和练习。相较于官方的文档，本书偏向实践，着眼于使用 hook 解决实际开发中的问题，除了讲解 hook 如何使用，还解释了为何这样使用，能帮助你更容易地将这些理论知识应用到自身的项目中。

本书的翻译由周轶、张兆阳和颜宇共同完成。三位译者均就职于国内知名的互联网公司，在前端领域深耕多年，在技术深度和广度上都有着丰富的积淀。其中第 6～8 章和第 10 章由译者周轶完成，第 1～4 章由译者张兆阳完成，其余章节由译者颜宇完成。本书知识覆盖面广，专业性较强，翻译此书我们如履如临，唯恐不能准确地还原作者本意，阐明技术细节。在译文中难免出现一些疏漏，希望各位读者不吝指正。

在此由衷地感谢清华大学出版社的各位编辑老师，是他们的专业能力和不懈的付出保障了本书的顺利出版。

周轶　张兆阳　颜宇

序　言

作为一名高中教师和程序员，我有得天独厚的条件可以为学校开发内部的教学类、学习类以及管理类应用程序。我了解学生和教职员工的第一手情况，以及他们的日常需求，可以与他们合作开发易于使用的应用程序和工具，帮助他们更轻松地规划、沟通、理解以及娱乐。我从编写答题应用程序和生词配对游戏开始，随后使用 jQuery 和模板技术开发了课程管理应用程序和服务预约应用程序。接下来，科学部门提出了为课程订购设备的要求，管理层希望实现员工发布公告的功能，而 ICT 的工程师们则希望能让员工上报软件问题和硬件故障，并可以管理这些问题和故障。我们可以分别开发一个座位预订系统、一个网站信息管理系统、一个定制化的在线日历、一个可互动的排班表，或者一个体育比赛日志系统，并且这些系统都采用统一的外观和风格，这个主意怎么样？

每个项目的需求各不相同，而其中又有很多重复和类似的功能，这些功能是可以在多个应用程序之间复用的。为了加快开发速度，我改用 Node.js、Express、Handlebars、Backbone 和 Marionette，使用 JavaScript 做全栈开发。尽管有时需求变更引起的改动会很棘手，特别是模型、视图和控制器之间的数据流转并不是很理想，但我可以用这种开发模式应付大多数情况。用户反馈不错，但我意识到代码存在问题，必须要面对并解决这些问题。

当我遇到 React 后，这一切的问题都迎刃而解了。好吧，有一些夸张了。但是 React 中的组件、prop、状态以及自动重新渲染令我印象深刻，这是之前其他框架不具备的。我一个接一个地将应用程序迁移到React。每完成一次迁移，都会令应用程序更容易理解和维护。通用组件可以被重用，因此我可以毫不犹豫地、快速地修改应用程序，为其添加新的功能。虽然我并不是 React 的狂热粉丝(我支持框架多样性)，但可以算是一个 React 的追随者，享受其带来的优秀的开发体验和良好的用户反馈。

随着 React Hooks的引入，我的代码更加简洁了。类组件生命周期方法中散落的代码被集中到函数式组件中，或者外部的自定义hook中。我们可以很容易地分割、维护以及共享特定功能的代码，包括设置文档标题、访问本地存储空间、管理上下文值、测量屏幕尺寸、订阅服务或者请求数据，并且可以更容易地使用第三方库的功能，例如 React Router、Redux、React Query 和 React Spring。React Hooks 为我们提供了一种新的理解 React 组件的方式，尽管在 React Hooks 刚推出时，你需要留心其中的一些问题，但是在我看来，这无疑是一个非常棒的功能。

Hooks 一定是 React 未来的发展方向之一。Concurrent 模式将会成为新的常态，时间切片(time-slicing)也会被加入 React 中，时间切片能够令渲染不再阻塞主线程，甚至当一个组件的 UI 正在被构建时，另一个组件仍可以直接渲染诸如用户输入的高优先级更新。选择性的“注水”

能够令 React 只在用户使用时才加载相关组件的代码，而 Suspense API 将会支持开发者在加载代码和资源时更精细地控制加载状态。

React 团队专注于提供更好的开发体验，从而帮助开发者开发出具有优秀用户体验的产品。React 仍然在持续迭代，最佳实践将会不断涌现，但我希望本书能够帮助你牢固地掌握现有功能，并为那些即将到来的令人兴奋的特性做好准备。

作 者 简 介

John Larsen 从 20 世纪 80 年代开始从事编程工作，最开始是在 Commodore VIC-20 上编写 Basic，随后又涉猎了 Java、PHP、C#以及 JavaScript 等领域。他还编写了同样由曼宁出版社出版的 *Get Programming with JavaScript* 一书。他在英国当了 25 年的数学老师，为高中生讲授计算机知识，并为学校开发与教学类、学习类以及沟通有关的 Web 程序。

致　　谢

通常此时我会感谢我的朋友和家人，感谢他们对我的耐心。因为我一直将自己与世隔绝，一直在疯狂地敲打着打字机的键盘，不断创作着，而其他人则还要继续正常的生活。但在 2020 年，有一些这样或者那样的事情发生，让我们的生活偏离了正常的轨道。因此，我想感谢所有那些在困难时帮助他人的朋友，无论他们的付出是多是少，都令周围人的生活更加美好。

感谢我在曼宁出版社的编辑，Helen Stergius，感谢她对我的耐心和鼓励。编写一本书是一个漫长的过程，但有了像 Helen 这样优秀编辑的支持和建议，令写书这件事轻松了许多。还要感谢 John Guthrie 和 Clive Harber，他们格外关注细节，并给出了诚恳且具有建设性的反馈，令本书中的代码和解释更加清晰和一致。我还要感谢我的制作编辑 Deirdre Hiam、文字编辑 Sharon Wilkey、校对 Keri Hales 以及审稿编辑 Aleksandar Dragosavljević。

感谢所有的审稿人：Annie Taylor Chen、Arnaud Castelltort、Bruno Sonnino、Chunxu Tang、Clive Harber、Daniel Couper、Edin Kapic、Gustavo Filipe Ramos Gomes、Isaac Wong、James Liu、Joe Justesen、Konstantinos Leimonis、Krzysztof Kamyczek、Rob Lacey、Rohit Sharma、Ronald Borman、Ryan Burrows、Ryan Huber 和 Sairam Sai，你们的建议使本书更加出色。

关于封面插图

本书封面插图的标题为“Femme de la Carie”，也被称为“来自卡里亚的女人”，选取自Jacques Grasset de Saint-Sauveur(1757—1810)所著的 *Costumes de Différents Pays*，该书于 1797 年在法国出版，是一本关于各国服饰的合集。其中每一页插图都是手工绘制和着色的，做工非常精致。Grasset de Saint-Sauveur 的藏品种类丰富，生动地提醒我们，仅仅在 200 年前，世界上的城镇和地区在文化上是多么的不同。那时人们彼此之间的沟通很少，说着不同的方言和语言。站在街头或乡村，仅从他们的衣着就很容易辨别他们来自何处、从事哪种职业以及生活状况如何。

从那时起，我们的着装已经发生了改变，如此丰富的多样性也逐渐消失了。现在很难区分不同大陆的居民，更不用说区分不同的城市、地区或国家的居民了。也许是我们用更多样化的个人生活，同样也是更加多变和快节奏的科技生活，取代了文化的多样性。

在当前这个很难区分各种计算机书籍的时代，曼宁出版社以两个多世纪前世界各地的生活多样性为基础，在本书的封面中重现了 Grasset de Saint-Sauveur 的作品，以体现计算机科技的创造性和主动性。

关 于 本 书

本书面向的读者是有经验的 React 开发者。本书介绍了目前 React 内置的 hook 特性，并展示了在开发应用程序时，如何在 React 函数式组件中使用 hook 管理组件内部状态，跨多组件管理状态，以及经过外部 API 同步组件状态。本书还演示了 hook 方法在封装、重用、简化组件代码、扩展性等方面的优势。同时，还探讨了 React 团队仍在研发的、更具实验性的 Suspense 和 Concurrent 模式。

本书读者对象

如果你之前使用过 React，并且希望了解 hook 究竟能如何改进代码，怎样将代码从类组件迁移到函数式组件，如何在代码中整合 Suspense 和 Concurrent 模式以提升开发体验和用户体验，那么本书将会为你解答这些疑问。你应该具备使用 create-react-app 新建应用程序，并使用 npm(或者 Yarn)安装相关依赖的基础。本书的示例代码采用了现代 JavaScript 语法和模式，例如解构、默认参数、扩展运算符、可选链运算符。因此，虽然这些语法在第一次被使用时会有简短的介绍，但是还是希望你对它们越熟悉越好。

本书的结构：路线图

本书包括两部分，共 13 章。

第 I 部分介绍了 React Hooks 的语法和使用方法，这是 React 内置的，非实验且稳定的最新特性。为你展示了如何开发自己的 hook，以及如何使用现有 React 第三方库提供的 hook。

- 第 1 章会从 React 最近的一些更新以及即将进行的一些改动开始讲述，尤其会关注 React Hooks 如何帮助你组织、维护以及分享组件。
- 第 2 章介绍第一个 hook——useState。组件可以使用 useState 管理状态，并在状态值发生变化时触发重新渲染。
- 有时候，多个状态之间会产生关联关系，其中一个状态变化会引起其他状态随之改变。第 3 章讲解的 useReducer hook 为你提供一种集中管理多个状态的方法。
- React 旨在令应用程序的状态与 UI 保持同步。在有些情况下，应用程序需要从其他地方获取状态，或者在文档之外显示状态，例如浏览器的标题中。当应用程序在组件范

围之外执行副作用操作时，应当将这部分代码用 useEffect hook 包装起来，以便同步所有的代码块。第 4 章详细讨论 useEffect。

- 第5章使用useRef hook在不引起重新渲染的情况下更新状态(如操作一个计时器的ID)以及维持页面中元素的引用，例如表单中的文本框。
- 应用程序由多个组件组成，第 6 章会带你研究多个组件之间共享状态的策略，如何通过prop向下传递状态；会向你展示如何共享useState和useReducer 的 updater(更新)函数，以及 useReducer 的 dispatch(分派)函数；并且还会展示如何利用 useCallback hook 为函数创建不会被改变的引用。
- 有时候，组件会依赖函数以某种方式生成或者转换数据。如果这些函数需要执行相对较长的时间才能结束，你就会希望只有在绝对必要时才调用它们。第 7 章将会展示如何利用 useMemo hook 限制大开销函数的执行。
- 同一个状态值有时候会在应用程序中为多个组件共用。第 8 章解释了如何避免在多级组件间自上而下传递 prop，使用 React Context API 以及 useContext hook 共享状态。
- React Hooks 仅仅是函数。你可以将调用这些 hook 的代码迁移到组件外的函数中。这类函数被称为自定义 hook，它们可以在组件之间共享，也可以跨项目共享。第 9 章解释了其中的原因，并通过大量的示例指导你创建自定义 hook，还将会重点阐述 hook 的规则。
- 主流的 React 库均已经升级到与 hook 协作的版本。第 10 章将会使用 React Router 的第三方 hook 管理 URL 中的状态，同时还使用 React Query 轻松地将 UI 与存储在服务器的状态保持同步。

本书的第Ⅱ部分解释大型应用程序如何更加有效地加载组件，以及如何使用 Suspense 组件和错误边界将回退 UI 作为加载资源管理。随后，深入探讨一些要用于整合 Suspense 与数据加载的实验性 API，最后则是那些在 Concurrent 模式下运行的实验性 API。

- 第 11 章讨论代码分割。我们利用 React.lazy 懒加载组件，当组件被懒加载时使用 Suspense 组件展示回退 UI，当加载过程发生错误时利用错误边界展示回退 UI。
- 在第 12 章中，我们将涉及更多的实验领域，看看如何将数据获取以及图片加载与 Suspense 整合在一起。
- 最终，在第 13 章中，我们将会探索那些只能在 Concurrent 模式下运行的实验性 API，包括 useTransition 和 useDeferredValue 这两个 hook 以及 SuspenseList 组件，它们都可以改进应用程序中状态变更时的用户体验。确切地说，它们还尚未确定最终的工作方式，但是本章将会提前讲解它们试图解决的问题。

虽然，本书的主要示例是随着书中课程的深入逐步创建起来的，但是这并不妨碍你直接学习某一章或者某一个 hook。如果你想单独运行某个代码示例，可以从代码仓库的对应分支中下载代码，然后运行即可。

上述章节还包括一系列的练习，便于我们实践所学的知识。这些练习大多数会让你将示例

应用程序中某一个页面的功能在另外一个页面重复实现。例如，书中的示例先演示如何更新Bookables页面，然后再要求你在Users页面中执行同样的操作。实践是一种有效的学习策略，不过在必要的情况下，也可以直接查看代码仓库中已经写好的代码。

关于代码

在本书中，随着每一章的深入，我们将逐步创建一个可以运行的预订应用程序，并将其作为本书的示例。这个示例为我们讨论以及实践React Hooks提供了良好的依托。但请注意，本书的重点在于hook，而非这个预订应用程序，因此本书列出了大部分的代码，然而仍然有一些代码更新后并没有同步在书中，你可以从示例应用程序在GitHub的代码仓库中找到最新的代码。代码仓库的地址是https://github.com/jrlarsen/react-hooks-in-action。

书中一些简单的示例并不属于预订应用程序的一部分。这些代码，如果是基于React的，就会保存在CodeSandbox中；如果是基于原生JavaScript的，则会被保存在JS Bin中。本书的代码清单中列出了指向GitHub、CodeSandbox或者JS Bin的对应链接。

本书的示例均已在React 17.0.1下经过了测试。除了第13章，因为其中的示例使用了实验版React，所以无法保证除所在分支使用的React版本之外的其他版本能够运行这部分的示例。

可扫描封底二维码获取本书网上资源。

目　录

第Ⅰ部分
React Hooks介绍及应用

本书的第Ⅰ部分将介绍 React Hooks 以及 React 17 第一个稳定版中的关键 hook。你可以从中学会如何在函数式组件中管理状态，如何与子组件和更深层级的子组件共享状态，如何将状态同步给外部服务、服务器和 API。还将学会如何创建自己的 hook(同时遵循规则)，并充分利用已建立的库中的第三方 hook，如 React Router、React Query 和 React Spring。

本部分将以一个预订应用程序作为示例进行讲解，从中你将学会如何加载和管理数据，如何编排组件间的交互，以及如何响应用户的操作。但首先，要弄清楚什么是 hook？为什么它们是迈向成功的一步？

第 1 章

逐渐演进的React

React 是一个用于构建漂亮用户界面的 JavaScript 库。React 团队期望为开发者提供尽可能完美的开发体验，以便激励开发者开发出受欢迎且高效的应用程序。本书针对 React 库中一些最新添加的内容提供了指南。这些最新内容能够简化代码，提高代码的复用性，并有助于使应用程序更灵活，响应速度更快，从而更好地造福开发者和用户。

本章将简要概述 React 及其最新特性，以期激发你对之后章节的兴趣。

1.1 什么是 React

假如你正在为 Web、桌面应用程序、智能手机，或虚拟现实(Virtual Reality，VR)创建用户界面(User Interface，UI)。你期望你的页面或者应用程序能够显示可随时间变化的各种数据，例如，经过认证的用户信息、可被筛选的商品列表、数据可视化，或用户详情。同时，你期望用户也可与应用程序进行交互，例如，选择过滤条件、数据集或用户，填写表单，或探索 VR 空间！此外，你的应用程序可能会消费来自网络或者互联网的数据，例如，社交媒体的更新、股票行情或产品的可用性。React 都将为你提供帮助。

React 使构建可组合且可复用的用户界面组件变得更简单，并且这些组件可以对数据变化和用户交互做出反应。以一个社交媒体网站的页面为例，该页面包含了按钮、帖子、评论、图像和视频，以及许多其他界面组件。当用户向下滚动页面、打开帖子、添加评论或切换到其他界面时，React 会帮助界面进行更新。此外，页面上的某些组件可能包含了重复的子组件，这些子组件包含了相同的元素结构，只是内容不同。而这些子组件也可以由组件构成！如缩略图、重复的按钮、可单击的文本和大量图标。总而言之，这个页面包含了上百个这类元素。但是，如果将如此丰富的界面拆分成多个可复用的组件，那么开发团队就可以更轻松地专注特定的功能领域，并将这些组件应用于多个页面。

React 的核心目标之一是简化组件的定义和复用，并将它们组合成复杂但易于理解且可用的接口。类似的还有其他的前端库(如 AngularJS、Vue.js 和 Ember.js)，但本书是一本关于 React

的书籍，因此我们将专注于 React 如何处理组件、数据流和代码复用。

在接下来的小节中，我们将深入了解 React 如何帮助开发者构建此类应用程序，重点介绍它的如下五个关键特性：

- 通过可复用和组合式的组件构建 UI。
- 使用 JSX 描述 UI——HTML 样式模板和 JavaScript 的混合。
- 在不引入太多惯用约束的情况下充分利用 JavaScript。
- 智能地同步状态和 UI。
- 帮助管理代码、资源和数据的加载。

1.1.1 用组件构建 UI

社交媒体网站丰富的、具有层级结构的多层级用户界面，均可借助 React 设计和编码完成。但是现在，我们先从一些相对简单的内容入手，感受一下 React 的特性。

假设你想构建一个答题应用程序，帮助学生检验学习情况。那么，你的组件应该能够控制提问的显示和隐藏，并且也可以控制答案的显示和隐藏。一套完整的提问与答案看上去应该如图 1.1 所示。

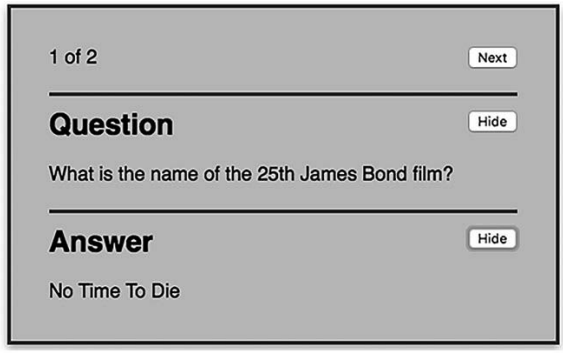

图 1.1 仅显示一个提问和一个答案的答题应用程序界面

你可以为提问区域和答案区域各创建一个组件。但两个组件的结构是一样的，每个组件都有一个标题、一些可显示或隐藏的文字，以及一个控制显示或隐藏的按钮。React 可以将其简化成一个组件的定义，例如 TextToggle 组件。你可以在提问和答案部分同时使用该组件，可将标题、文字和文字的显示与否传递给每个 TextToggle 组件。所传递的值将作为属性(或 prop)，如下面的代码所示：

```
TextToggle title="Question" text="Who created JavaScript?" show={true} />

<TextToggle title="Answer" text="Brendan Eich" show={false} />
```

等一下！这是什么代码？是 HTML？XML？JavaScript？其实，编写 React 代码就是编写 JavaScript 代码。但是 React 为 UI 描述提供了一个类似于 HTML 语句的方式——JSX。在运行

应用程序之前，需要对 JSX 进行预处理，将其转换为创建用户界面元素的真实的 JavaScript。最初，你会觉得将 HTML 混入 JavaScript 似乎有一点奇怪，但事实证明，方便是一个很大的优势。一旦你的代码最终在浏览器(或其他环境)中运行，它就真的只是 JavaScript。Babel 包可以将编写的代码编译成要运行的代码。可在 https://babeljs.io 找到更多关于 Babel 包的信息。

本章仅提供 React 的简要概述，因此在这里不会深入探讨 JSX。不过，它还是值得一提的，因为它是 React 开发中广泛使用的一部分。实际上，在我看来，React 的 JavaScript 特性正是它的吸引力之一——虽然也有其他意见——但在大多数情况下，它并没有引入太多约束。尽管学无止境，但成为一名优秀的 JavaScript 程序员和成为一名优秀的 React 程序员所需要具备的技能是非常相似的。

如果你已经创建了 TextToggle 组件，那么下一步需要做什么呢？通过 React，你可以利用已有的组件定义一个新组件。可以将提问卡片封装成自己的 QuestionCard 组件，显示提问与答案。如果想一次显示多个提问，就可以将多个 QuestionCard 组件组成一个 Quiz 组件 UI。

图 1.2 显示了由两个 QuestionCard 组件组成的 Quiz 组件。Quiz 组件仅仅是 QuestionCard 的容器，除了它包含的卡片外没有其他用户可见的部分。

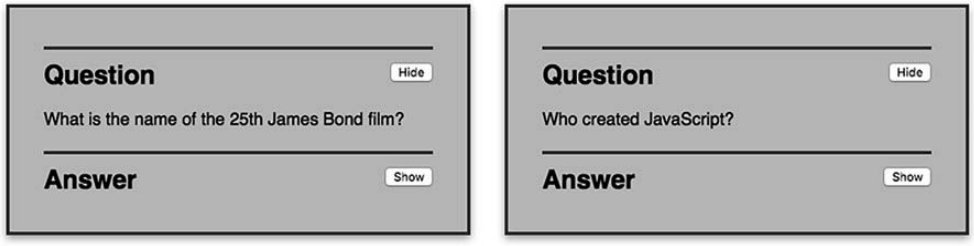

图 1.2　显示包含两个 QuestionCard 组件的 Quiz 组件

由此可见，该 Quiz 组件是由 QuestionCard 组件组成的，QuestionCard 组件是由 TextToggle 组件组成的，而 TextToggle 组件是由标准的 HTML 元素构成的——如 h2、p 和 Button。最终，Quiz 组件包含了全部的原生 UI 元素。图 1.3 显示了 Quiz 组件简单的层级结构。

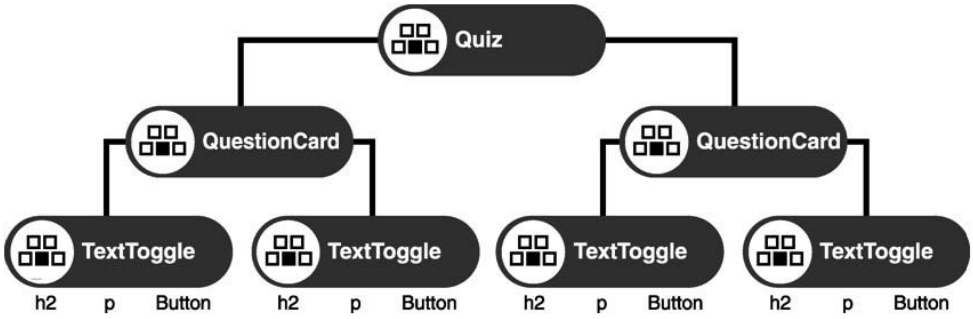

图 1.3　Quiz 组件的层级结构

React 使该组件的创建和组合变得更容易。而且，一旦你制作了自己的组件，就可以轻松

地复用和共享该组件。你可以想象一个学习资源网站，其中包含不同的页面，每个页面都针对不同的话题。可以在每个页面上都应用你的Quiz 组件，再向各个主题传递答题应用程序的数据即可。

可以从诸如 npm 的包管理仓库下载许多 React 组件。当下拉菜单、日期选择器、富文本编辑器或者其他可能的答题应用程序所需的模板已经过良好的测试和实践，放置备用时，将不必重新创建符合常见场景的组件，无论这些组件是复杂还是简单。

React 为应用程序到组件间的数据传递提供了机制和模式。实际上，状态与 UI 的同步是 React 的核心，也是 React 的用武之地。

1.1.2 同步状态和 UI

React 保持着应用程序 UI 与数据的同步。应用程序在任何时候持有的数据，都被称为应用程序的状态，这些数据可能是当前的帖子、登录用户的详细信息、显示或隐藏评论，或者文本输入字段的内容。如果有新的数据通过网络到达应用程序，或用户通过按钮或文本输入框更新某个值，React 都会计算出需要进行哪些修改以用于显示，并有效地更新它。

React 会智能地编排更新的顺序和时机，以优化应用程序的感知性能，改善用户体验。图 1.4 显示了该想法，即 React 通过重新渲染 UI 来响应组件状态的变化。

但是更新状态和重新渲染并不是一次性的任务。应用程序的访客很可能引起大量的状态改变，而 React 为了呈现拥有最新状态值的 UI，需要反复请求组件。而组件的工作是将状态和 prop(传到组件中的属性)转变为用户界面描述。然后 React 将获取这些 UI 描述，并在必要时对浏览器的文档对象模型(Document Object Model，DOM)进行更新。

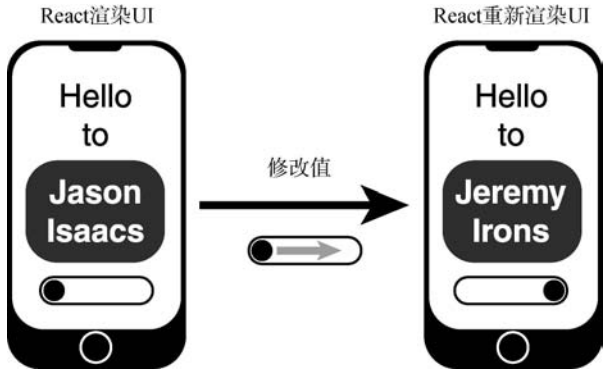

图 1.4 当组件的状态值改变时，React 会重新渲染 UI

循环图

为了表示状态变更和 UI 更新的持续循环，本书使用循环图说明组件和 React 之间的交互。图 1.5 是一个简单示例，显示了当组件第一次出现时，以及用户更新值时，React 是如何调用组件代码的。

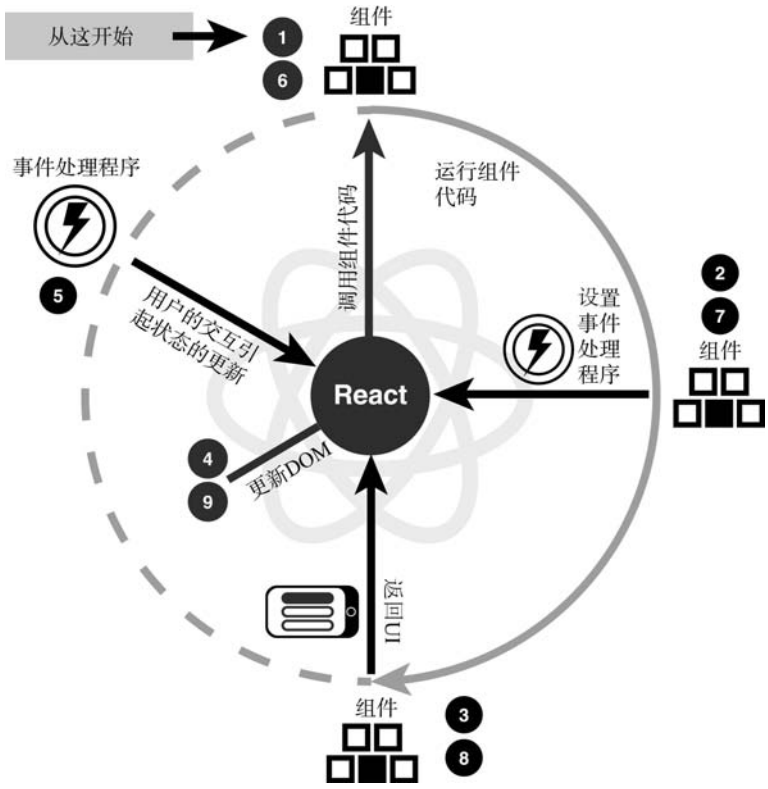

图 1.5　React 调用或重新调用组件，生成使用最新状态值的 UI 描述

循环图附有表格，如表 1.1 所示，其更详细地描述了图表的步骤。该图和表格并没有列出所有事件，但已列出所有关键步骤，以帮助你理解组件在不同场景中工作方式的相同点与不同点。

例如，图 1.5 并没有展示 React 中的事件处理程序是如何更新状态的；相关内容会在后面介绍相关 React Hooks 的图表中详细讲解。

表 1.1　React 调用和重新调用函数式组件时的关键步骤

步骤	发生了什么	讨论
1	React 调用组件	为了达到生成页面 UI 的目的，React 会遍历组件树，并调用每个组件。React将向每个组件传递被设置为 JSX 属性的 prop
2	组件指定一个事件处理程序	例如，事件处理程序可以监听用户单击、计时器的触发或资源加载。处理程序会在后续运行时更改状态。当 React 在步骤 4 中更新 DOM 时，它会将处理程序 hook 到 DOM
3	组件返回它的 UI	组件使用当前状态值生成用户界面并返回该用户界面，即完成工作

(续表)

步骤	发生了什么	讨论
4	React 更新 DOM	React将组件返回的UI描述与应用程序当前的UI描述进行比较。其将高效地对DOM 进行必要的更改，并根据需要，设置或更新事件处理程序
5	触发事件处理程序	一个事件的发生，将会触发处理程序的运行。处理程序将会更改状态
6	React 调用组件	React 得知状态值已经变更，因此必须重新计算 UI
7	组件指定一个事件处理程序	这是新版本的处理程序，并且使用了更新后的状态值
8	组件返回它的 UI	组件使用当前的状态值生成用户界面并返回用户界面，即完成工作
9	React 更新 DOM	React将组件返回的UI描述与应用程序之前的UI描述进行比较。它将高效地对 DOM 进行必要的更改，并根据需要，设置或更新事件处理程序

插图将使用一致的图标来表示关键的对象和插图文本讨论的动作，如组件、状态值、事件处理程序和 UI。

答题应用程序中的状态

就像本章开头讨论的那样，社交媒体页面通常需要很多状态，例如，加载新的帖子、用户为喜欢的帖子点赞、添加评论，以及以各种方式与组件交互。部分上述状态(如当前用户)可能会在许多个组件之间共享，而其他状态(如评论)可能仅应用于帖子本身。

答题应用程序有一个提问与答案组件—— QuestionCard，如图 1.6 所示。用户可以显示或隐藏提问或者答案，并且可以选择进入下一个提问。

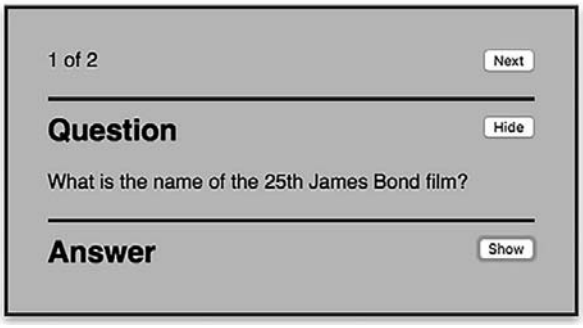

图 1.6 答案被隐藏的提问与答案组件

该 QuestionCard 组件的状态包括显示当前提问与答案所需的信息：

- 提问的序号。

- 提问的总数。
- 提问的文字。
- 答案的文字。
- 提问被隐藏还是被显示。
- 答案被隐藏还是被显示。

单击答案的 Show 按钮会改变组件的状态。可能会有一个 isAnswerShown 变量从 false 变为 true。React 将会注意到状态已发生变化，并更新被显示的组件，以显示答案文本，且将按钮的文本从 Show 切换到 Hide(见图 1.7)。

单击 Next 按钮将会改变提问的序号。它将从提问 1 切换到提问 2，如图 1.8 所示。如果整个答题应用程序的提问与答案都在内存中，那么 React 可以立即更新显示；如果需要从文件或者服务中加载它们，那么 React 可以在更新 UI 之前等待数据被获取；或者，如果网络很慢，则会显示一个加载的标识，如旋转标志。

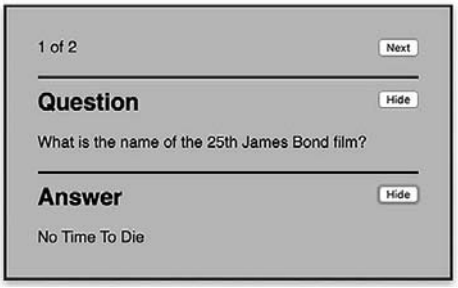

图 1.7　显示答案的提问与答案组件

图 1.8　显示第二个提问的提问与答案组件。该答案被隐藏

这个简单的答题应用程序示例并不需要太多状态履行职责。大多数真实世界的应用程序都会更加复杂。决定状态应该保存在哪里——组件是否应该管理它自己的状态，组件是否应该共享状态，以及状态是否应该全局共享——这些决定是构建应用程序的重要组成部分。React 为这三种场景都提供了机制，并且已发布的包(如 Redux、MobX、React Query 和 Apollo Client)都提供了管理状态的方法，这些状态数据会被存储在组件之外。

以往，组件是否管理自己的某些状态决定了你将使用哪种组件创建方法；React 提供了两种主要方法：函数式组件和类组件，将在接下来的章节讨论这些内容。

1.1.3　理解组件的类型

React 允许使用以下两种“JavaScript 结构”定义组件：函数和类。在 React Hooks 之前，当组件不需要任何本地状态时，可以使用一个函数(在此通过 props 将其所有数据传递给该函数)：

```
function MyComponent (props) {
   // Maybe work with the props in some way.
   // Return the UI incorporating prop values.
}
```

当组件需要管理自己的状态、执行副作用(如加载其数据或体验 DOM)或直接响应事件时，需要使用类组件：

```
class MyComponent extends React.Component {
  constructor (props) {
    super(props);

    this.state = {
      // Set up state here.
    };
}

componentDidMount () {
  // Perform a side effect like loading data.
}
  render () {
    // Return the UI using prop values and state.
  }
}
```

类组件在构造函数中设置其状态

类组件可以包括其生命周期各个阶段的方法

类组件有一个 render 方法，用于返回 UI

添加 React Hooks 意味着现在可以使用函数式组件管理状态和副作用：

```
function MyComponent (props) {
  // Use local state.
  const [value, setValue] = useState(initialValue);
  const [state, dispatch] = useReducer(reducer, initialState);

  useEffect(() => {
    // Perform side effect.
  });

  return (
    <p>{value} and {state.message}</p>
  );
}
```

使用 hook 管理状态

使用 hook 管理副作用

从函数中直接返回 UI

React 团队推荐在新项目中使用函数式组件(它们还没有计划移除类组件，因此并不需要大范围地重写已有项目)。表 1.2 列举了组件类型和说明。

表 1.2 组件类型和说明

组件类型	说明
无状态函数式组件	传递属性并返回 UI 的 JavaScript 函数
函数式组件	传递属性，并使用 hook 管理状态和执行副作用，以及返回 UI 的 JavaScript 函数
类组件	包含一个用于返回 UI 的 render 方法的 JavaScript 类。它可以在其构造函数中设置状态，并在其生命周期方法中管理状态和执行副作用

函数式组件仅返回用户界面的 JavaScript 函数。在编写组件时，开发者通常使用 JSX 详述 UI。而 UI 可能取决于传递给函数的属性。有了无状态的函数式组件，故事就结束了；它们将

属性转换为 UI。更通俗地讲，函数式组件可以包含状态和副作用。

类组件是使用 JavaScript 类语法构建的，并从 React.Component 或 React.PureComponent 基类扩展而来。它们拥有一个可初始化状态的构造函数，以及一些可以作为组件部分生命周期的方法。例如，当 DOM 被更新为最新的组件 UI 时，或传递给组件的属性发生更改时，React 可以调用这些生命周期。类组件也拥有一个返回组件 UI 描述的 render 方法。类组件是构建有状态组件的一种方式，这种方式可以引发副作用。

我们将在 1.3 节学习，相比于类组件，应用了 hook 的函数式组件如何更好地提供一种方式，来构建有状态组件和管理副作用。首先，让我们更全面地了解 React 中的新功能，以及这些新功能如何与 React 一起更好地协同工作。

组件的副作用

通常 React 组件会将状态转换为 UI。当组件代码在主要关注点以外执行操作时——可能是从网络中获取博客文章或股票价格，设置在线服务的订阅，或通过直接与 DOM 交互来关注表单字段，或测量元素的尺寸——我们描述的这些动作是组件的副作用。

我们希望应用程序及其组件的行为是可预测的，因此应确保任何必要的副作用都是经过深思熟虑，并且是明显可见的。如第 4 章所述，React 提供 useEffect hook 来帮助我们在函数式组件中设置和管理副作用。

1.2　React 中的新增功能

React 16 包括对核心功能的重写，以此为稳定推出新功能和新方法铺平道路。我们将在接下来的章节中探索几个最新的功能，其中包括：

- 有状态的函数式组件(useState，useReducer)
- 上下文的 API(useContext)
- 更清晰简洁的副作用管理(useEffect)
- 简单有效的代码复用模式(自定义 hook)
- 代码分割(lazy)
- 更快的初始化加载和智能渲染(Concurrent 模式——实验阶段)
- 更好的状态加载反馈(Suspense，useTransition)
- 更强大的调试、检查和剖析(开发工具和 Profiler)
- 有针对性的错误处理(错误边界)

以 use 开头的词——useState、useReducer、useContext、useEffect 和 useTransition——这些都是 React Hooks 的例子。可以在 React 函数式组件中调用这些函数，这些函数 hook 了关键的 React 功能点：状态、生命周期和上下文。React Hooks 允许向函数式组件、简洁封装的副作用添加状态，并在整个项目中复用代码。使用 hook 后将不再需要类，这可以使代码更精简、更健壮，从而更优雅。1.3 节将更详细地讨论 React 组件和 hook。

Concurrent 模式和 Suspense 提供了一种方法，可以帮助你更加慎重地考虑何时加载代码、数据和资源，并以更协调的方式处理状态的加载和内容的回退，例如在加载时展示一个旋转标志。目的是在应用程序状态的加载变化时改善用户体验，同时也改善开发者体验，使开发者更容易 hook 这些新行为。React 可以暂停那些开销很大但不紧急的组件渲染工作，并切换到其他紧急的任务，例如对用户交互做出反应，以此保持应用程序可及时作出响应，为用户提供平滑的体验。

https://reactjs.org 上的 React 文档是一个很好的资源，它对 React 哲学、API 和推荐使用的库提供了清晰的解释，并且结构良好。该文档也包含来自团队的博客文章和在线代码示例的链接，关于新特性的会议讨论，以及其他与 React 相关的资源。本书将专注 hook、Suspense 和 Concurrent 模式，你可查看官方文档了解更多有关 React 其他附加功能的信息。

1.3 可以为函数式组件添加状态的 React Hooks

正如 1.1.2 节所述，React 的核心优势之一是其如何将应用程序和组件的状态同步到 UI。例如，基于用户交互或系统和网络的数据更新而引起的状态变化，React 将智能高效地计算出应该对浏览器中(或其他环境中)的 DOM 或 UI 进行哪些更改。

组件本地的状态可以提升到树中更高的组件中，并通过属性在兄弟组件之间共享，或通过 React 的上下文机制或者高阶组件(一些组件作为参数，并返回一个包含被传入组件及额外功能的新组件的函数)在全局共享。对于一个包含状态的组件，过去使用继承自 React.Component 的 JavaScript 类的类组件。而现在，通过 React Hooks 即可为函数式组件添加状态。

1.3.1 有状态的函数式组件：更少的代码，更好的组织结构

与类组件相比，采用 hook 的函数式组件鼓励更简洁、更精炼的代码，这更有助于测试、维护和复用。函数式组件是返回了用户界面描述的 JavaScript 函数。该用户界面取决于传入的属性以及由组件管理的状态或通过组件访问的状态。图 1.9 是一个描绘函数式组件的图解。

该图显示了一个执行多个副作用的 Quiz 函数式组件：

- 加载提问数据——当用户选择新的提问集时的初始数据和新的提问数据。
- 订阅用户服务——该服务提供了当前在线的其他用户的最新信息，以便用户可以加入团队或挑战竞争对手。

在 JavaScript 中，函数可以包含其他函数，因此组件可以包含用于响应用户与 UI 交互的事件处理程序。例如，显示、隐藏或者提交答案，或是下一题。在组件内可以轻松封装副作用，例如获取提问数据或订阅用户服务。还可以在组件内涵盖用于清理这些副作用的代码，例如，取消任何未完成的数据获取和从用户服务中取消订阅。可以使用hook 将这些功能提取到组件之外，放入它们自己的函数中，以备重用或共享。以下是使用新的函数式组件方式的优点，而不是基于旧的类的方式。

- 更少的代码。
- 更好的代码组织——将与之相关的清理代码组织在一起。
- 可将功能提取到有利于重用和共享的外部函数。
- 更易于测试的组件。
- 不再需要像在类组件的构造函数中一样调用 super()。
- 不再需要 this 和 bind 函数。
- 更简单的生命周期模型。
- 本地状态处于处理程序、副作用函数和返回 UI 的作用域内。

图 1.9　带有状态并且封装了数据加载和管理服务订阅的 Quiz 函数式组件

列表中的所有项都有助于编写更易于理解、更易于使用和维护的代码。这并不是说细微的差别不会让第一次使用新方式的开发者陷入困境。本书将在更深入地研究每个概念及其新旧方式的关联时强调这些细微差别。

本书概述了构建组件的函数式方法，而不是使用类的方式。但为了激励使用新方法，有时值得将新方法与旧方法进行比较，因为发掘差异是很有趣的(对于 hook 来说，有点酷！)。即使你是 React 的新用户，并且从未见过类组件的代码也不必担心。因为本书后部使用的函数式组件是未来的首选方法。接下来的讨论仍有助于你理解这种新方法是如何简化和组织创建 React 组件所需的代码的。

本节的标题是“有状态的函数式组件：更少的代码，更好的组织结构”。然而，比什么更好？对于类组件而言，状态是在构造函数中设置的，事件处理程序被绑定到 this 变量上，副作用代码被拆分到多个生命周期方法中(componentDidMount、componentWillUnmount、componentWillUpdate 等)。因此，不同副作用和功能的相关代码在生命周期方法中并排放置很常见。在图 1.10 中可看到，用于加载提问数据和订阅用户服务的 Quiz 类组件代码是如何被拆分到多个方法之中，以及一些方法是如何混合用于这两个任务的代码的。

而采用了 hook 的函数式组件不再需要所有的生命周期方法，因为副作用可以被封装进 hook 中。这个改变使得代码更整洁，拥有更好的组织结构，正如图 1.10 中的 Quiz 函数式组件。将两个副作用分开，并将每个副作用的代码合并到一个地方，使代码的组织更加合理。改进后的组织结构可以更轻松地找到特定效果的代码、查看组件的工作方式，将来也更容易对其进行维护。事实上，将相同功能或相同副作用的代码放置于同一个地方更易于将它提取到外部函数中，这就是接下来要讨论的内容。

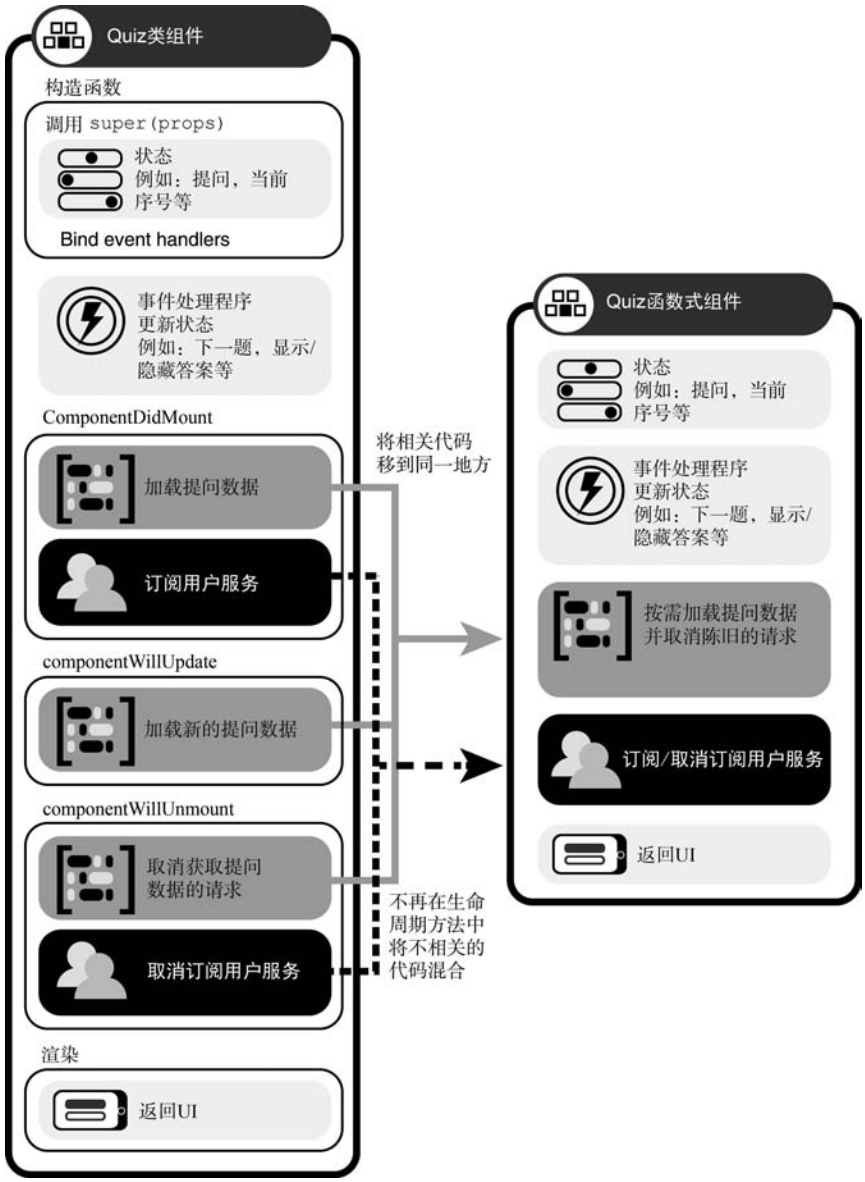

图 1.10 类组件的代码分布在各个生命周期方法中，而函数式组件的代码更少，组织结构更好但功能相同

1.3.2 自定义 hook：更易于代码复用

应用了 hook 的函数式组件鼓励将相关联的副作用逻辑放置于相同位置。如果某个副作用是许多组件都需要的功能，则可以进一步改善组织结构，将代码提取到属于该副作用的外部函数中，这时就可以创建所谓的自定义 hook。

图 1.11 显示了 Quiz 函数式组件的提问加载和用户服务的订阅任务是如何迁移到它们的自

定义 hook 中的。任何单独用于这些任务的状态都可以迁移到相应的 hook 中。

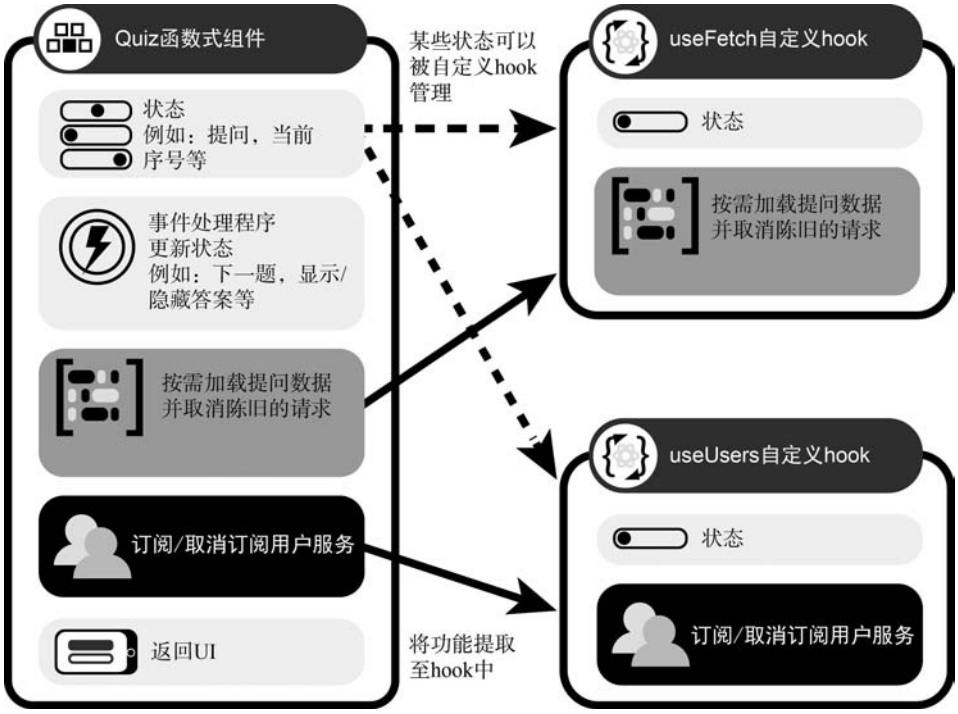

图 1.11　用于获取提问数据和订阅用户服务的代码可以提取到自定义 hook 中。伴随的状态也可以由 hook 管理

这并不是魔法。这就是函数通常在 JavaScript 中的工作方式：从组件中提取函数，然后在组件中调用。一旦有了自定义 hook，就不会被限制于仅在原始组件中调用它。可以在许多组件中使用它，与团队共享它，或将其发布供他人使用。

图 1.12 显示了一个新的、经过瘦身的，并使用 useUsers 和 useFetch 自定义 hook 分别执行用户服务订阅和提问获取任务的 Quiz 函数式组件，这些任务以前由组件自己执行。现在，第二个组件 Chat，也使用了 useUsers 自定义 hook。hook 使得这类功能在 React 中的共享变得更容易；可以在应用程序中的任何有需要的地方导入和使用自定义 hook。

每个自定义 hook 都可以维护自己的状态，无论该自定义 hook 需要执行怎样的职责。并且，因为 hook 仅仅是函数，所以，不论组件需要访问 hook 的任何状态，hook 都可以在其返回值中包含该状态。例如，为指定 ID 获取用户信息的自定义 hook 可以将获取的用户数据存储在本地，并将其返回给调用该 hook 的任何组件。每次的 hook 调用都将封装其自己的状态，就像其他函数一样。

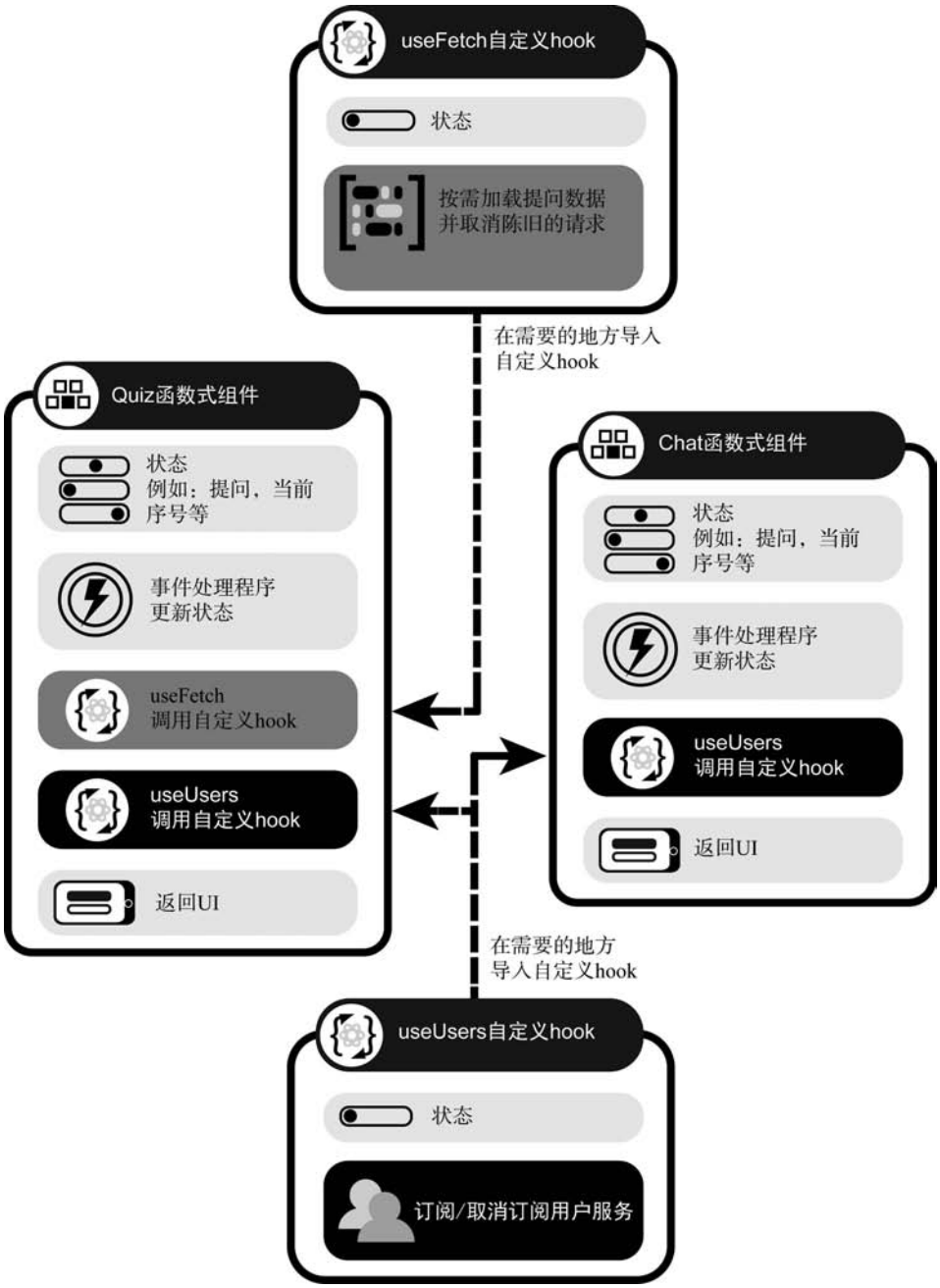

图 1.12 可以将代码提取到自定义 hook 中以供复用和共享。Quiz 组件同时调用 useUsers hook 和 useFetch hook。Chat 组件调用 useUsers hook

要了解更多已经被程序员抽象为自定义 hook 的各种常见功能，请访问 https://usehooks.com 的 useHooks 网站(见图 1.13)。

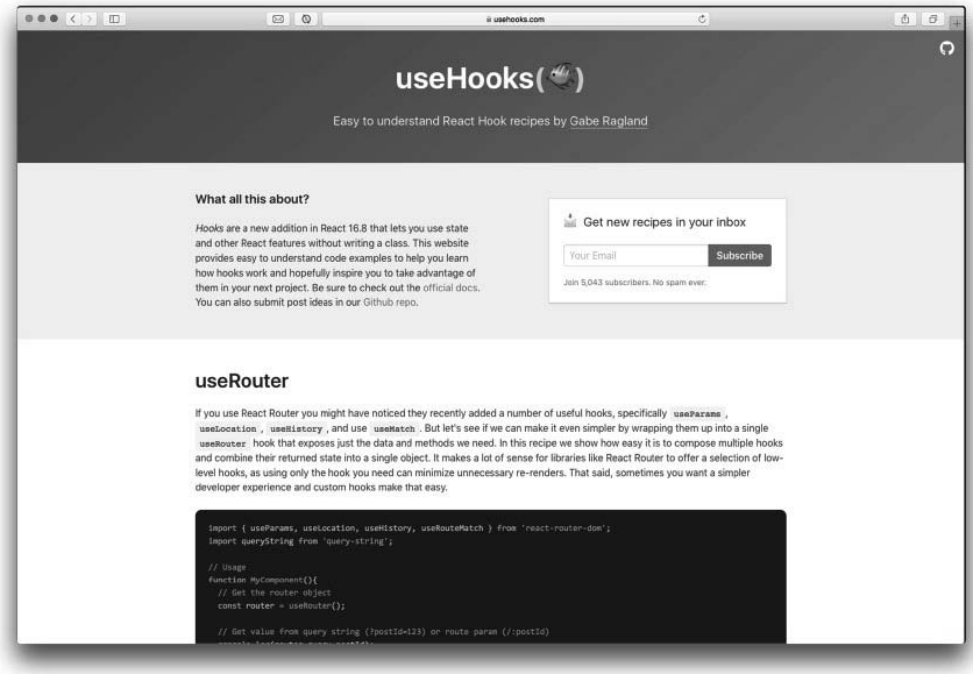

图 1.13　useHooks 网站有许多自定义 hook 的案例

该图显示了一些易于使用的自定义 hook：

- useRouter ——包装 React Router 提供的新 hook。
- useAuth——能够使任何组件获得当前的 auth 状态，并在当前 auth 状态改变时进行重新渲染。
- useEventListener——抽象了向组件添加和移除事件监听器的过程。
- useMedia——简化组件逻辑中进行的媒体查询。

在创建自己的 hook 之前，有必要搜索是否已存在满足场景的 hook，例如在 useHooks 网站或包管理仓库 npm 进行搜索。如果已为一些常见场景使用了库或框架，例如数据获取或状态管理，就请检查最新版本，查阅它们是否引入了 hook，以便更轻松地使用它们。

1.3.3　第三方的 hook 提供了完备的、经过良好测试的功能

跨组件共享功能并不新鲜；一段时间以来，它一直是 React 开发的重要组成部分。与旧的高阶组件和渲染 prop 相比，hook 提供了一种更清晰的代码共享方式和 hook 相关功能的方式，而高阶组件和渲染 prop 往往会导致代码的频繁嵌套(“封装地狱”)和错误层次结构的代码。

为了充分利用 hook 中更简单的 API 和更直截了当的集成方法，与 React 配合使用的第三方库已快速发布了新版本。本节将简要介绍以下三个示例：

- 用于页面导航的 React Router。

- 用于应用程序数据存储的 Redux。
- 用于动画的 React Spring。

React Router

React Router 提供了组件，该组件帮助开发者管理在应用程序页面间的导航。它的自定义 hook 可以轻松访问导航中涉及的常见对象：useHistory、useLocation、useParams 和 useRouteMatch。例如，useParams 允许访问页面 URL 中任何匹配的参数：

```
URL: /quiz/: title/: qnum
code: const {title, qnum} = useParams();
```

Redux

对于某些应用程序，单独的状态存储(store)可能才是合适的。Redux是一个流行的、用于创建此类状态存储的库，它通常通过 React Redux 库与 React 结合使用。从 7.1 版本开始，React Redux 提供了 hook，例如 useSelector、useDispatch 和 useStore，这些自定义 hook 使得 React 与 store 的交互更容易。例如，useDispatch 支持通过 dispatch action 更新 store 中的状态。假设已构建一个答题应用程序，该应用程序包含一个提问集，而你想添加一个提问：

```
const dispatch = useDispatch();
dispatch({type: "add question", payload: /* question data */});
```

新的自定义 hook 删除了一些用于连接 React 应用程序和 Redux store 的样板代码。React 也有一个内置的 hook——useReducer，它可为 dispatch action 提供一个更简单的模型，用于更新状态，并在某些情况下消除感官上对 Redux 的需求。

React Spring

React Spring 是一个基于 Spring 的动画库，目前提供五个 hook 来访问其功能：useSpring、useSprings、useTrail、useTransition 和 useChain。例如，要在两个值之间设置动画，可以选择 useSpring：

```
const props = useSpring({opacity: 1, from: {opacity: 0}});
```

React Hooks 使库作者可以更轻松地为开发者提供更简单的 API，这些 API 不会因组件层次结构潜在的深层嵌套的错误而使代码混乱。同样，一些其他的新 React 特性，例如 Concurrent 模式和 Suspense，可帮助库作者和应用程序开发者更好地管理其代码中的异步流程，并提供更流畅、响应更快的用户体验。

1.4 通过 Concurrent 模式和 Suspense 获得更好的 UX

我们希望为用户提供优秀的体验，帮助他们顺利并愉快地与应用程序交互。这可能意味着

他们可以在应用程序中高效地完成工作，在社交平台上与朋友联系，或在游戏中捕捉水晶。无论他们的目标是什么，我们设计和编码的接口应该是达到目的的手段，而不是绊脚石。他们在使用应用程序时，会在页面之间快速切换、滚动和点击，应用程序可能需要加载大量的代码，获取大量的数据，并尝试操作数据来提供用户需要的信息。

在版本 16 和 17 中重写 React 的很大一部分动机是为了构建可应对用户界面上的多重诉求的架构，如在用户持续与应用程序交互时加载和操作数据。Concurrent 模式是新架构的核心模式，Suspense 组件自然也满足了新的模式。但是它们解决了什么问题呢？

假设有一个应用程序，该应用程序可在一个长列表中显示产品，并且有一个供用户输入条件过滤列表的文本框。该应用程序会根据用户的输入更新列表。每次的按键输入都会触发代码对列表重新过滤，并需要 React 将更新后的列表组件绘制在屏幕上。大开销的过滤过程以及 UI 的重新计算和更新会占用处理时间，从而降低文本框的响应能力。而用户体验到的则是一种滞后的、缓慢的文本框，无法在键入时即时显示文本。显然是浏览器的代码运行编排导致了这种不完美的呈现，图 1.14 描述了这个问题，即长时间运行的操作会减慢屏幕的更新速度，从而导致较差的用户体验。

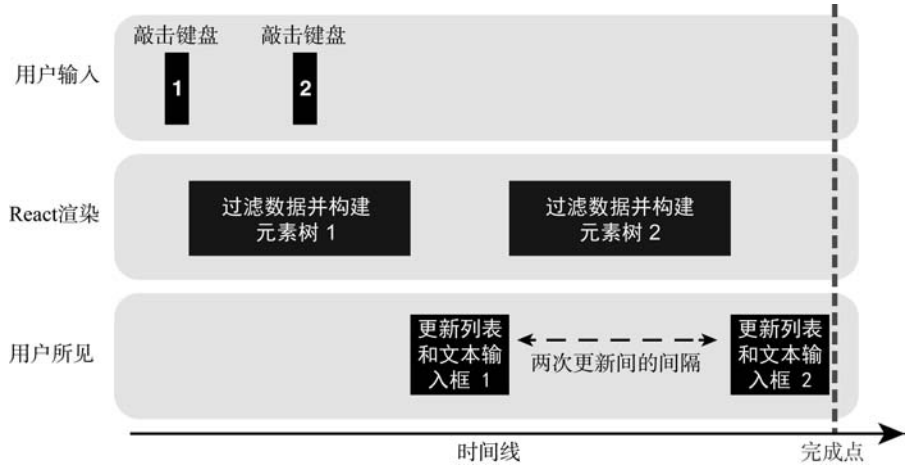

图 1.14　没有 Concurrent 模式的情况下，长时间运行的更新会阻塞诸如键盘敲击的交互

如果应用程序可以暂停和重新开始这些由键入引起的过滤任务，优先处理文本框的更新，以保证用户体验的流畅，那么这不是很好吗？向 Concurrent 模式问好！

1.4.1　Concurrent 模式

使用 Concurrent 模式后，React 能够以更细粒度的方式编排任务，暂停其构建元素的工作，检查差异，并为先前全部的状态变更更新 DOM，以确保对用户交互的响应。在之前的过滤应用程序示例中，React可以暂停被过滤后的列表的渲染，以确保用户输入的文本出现在文本框中。

Concurrent 模式如何实现这种魔法？新的 React 架构将其任务拆分为更小的工作单元，并

为浏览器或操作系统提供规则的介入点，以通知应用程序用户正在尝试与其交互。React 的调度器可以根据每个任务的优先级决定要做什么。可以暂停或放弃那些对组件树的部分协调和更改，以确保首先更新拥有更高优先级的组件，如图 1.15 所示。

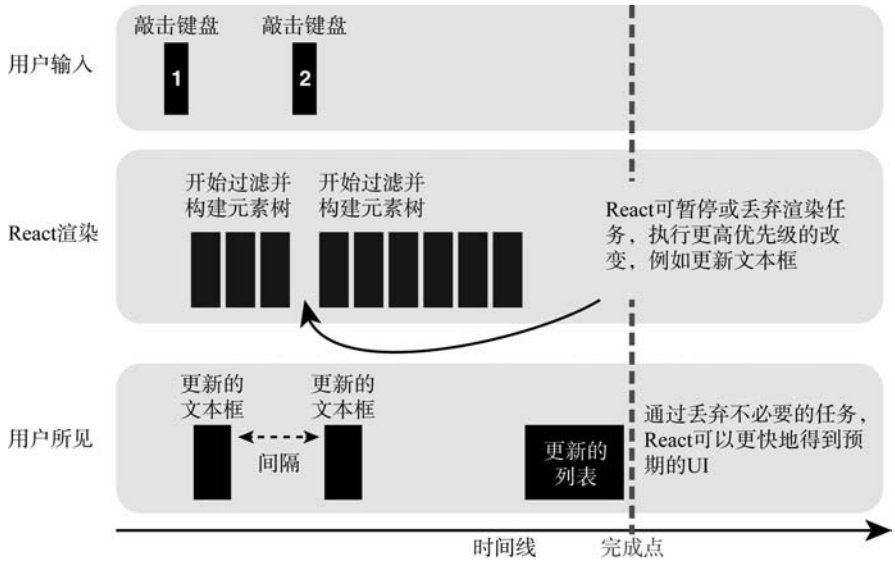

图 1.15 在 Concurrent 模式中，React 可以暂停长时间运行的更新，从而快速地响应用户交互

能够从这种智能调度中受益的不仅仅是用户交互。那些对传入数据的响应、延迟加载的组件或媒体，或其他异步进程也可以拥有更流畅的用户界面改善。当 React 在内存中为更新后的状态渲染 UI 时，可以持续显示目前的 UI，并保证用户与目前的 UI 交互(而不是加载中的旋转标志)。然后在其准备就绪时，再切换到新 UI。Concurrent 模式启用了几个新的 hook——useTransition 和 useDeferredValue，这些 hook 改善了用户体验，使得从一种视图到另一种视图，或从一种状态到另一种状态的改变变得平滑。它还可以与 Suspense 协同使用，即一个用于渲染回退内容的组件，一个用于明确某组件正在处于等待状态的机制，例如数据加载。

1.4.2 Suspense

如你所见，React 应用程序是由树状层级结构的组件构成的。为了在屏幕上显示应用程序的当前状态(如利用 DOM 来显示)，React 会遍历组件，并在内存中创建元素树和预期的 UI 描述。接着，React 会将最新的树与之前的树进行比较，并智能地决定需要进行哪些 DOM 更新以实现预期的 UI。Concurrent 模式会让 React 暂停处理部分元素树，其原因可能是为了处理更高优先级的任务，或是因为当前组件还没有准备好进行处理。

现在，采用了 Suspense 构建的组件可以在未准备好返回 UI 时挂起(注意，组件要么是函数，要么具有 render 方法并将属性和状态转换为 UI)。它们可能是正在等待组件代码、资源或数据的加载，还没有完整获取其描述 UI 所需的信息。React 可以暂停处理被挂起的组件，并继续遍

历元素树。但这在屏幕上看起来会是怎样？你的用户界面有漏洞吗？

除了挂起指定组件的机制，React 还提供了一个 Suspense 组件，可以使用它堵住挂起组件在用户界面中留下的漏洞。将 UI 包装在 Suspense 组件中，并利用该组件的 fallback(回退)属性告知 React，在一个或多个被包装的组件被挂起时要显示什么内容。

```
<Suspense fallback={<MySpinner />}>
  <MyFirstComponent />
  <MySecondComponent />
</Suspense>
```

Suspense 允许开发者有意识地管理多个组件的加载状态，或为单个组件、多个组件或整个应用程序进行回退显示。它为库作者提供了一种机制，将他们的 API 更新为支持使用 Suspense 组件的 API，这样他们的异步功能就可以充分利用 Suspense 提供的加载状态管理。

1.5　全新的 React 发布渠道

为了使应用程序的开发者和库作者能够在产出中充分利用稳定的功能，同时仍为即将推出的功能做好准备，React 团队已开始在不同的渠道发布代码：

- Latest(最新版)——稳定的 semver 版本发布。
- Next(下一版)——追踪 React 开发的 main 分支。
- Experimental(实验版)——包含实验阶段的 API 与功能。

对于生产环境，开发者应该紧跟最新发布的版本。最新发布的版本正是你从 npm(或其他包管理器)安装 React 时使用的版本。在撰写本书时，许多用于数据获取的 Concurrent 模式和 Suspense 都还在实验阶段。两个技术还在筹备中，API 可能会发生变化。React 和 Relay(用于数据获取)团队已经在新的 Facebook 网站上使用一些实验性功能，并已持续了一段时间。这种主动的使用有助于他们在上下文和一定范围内深入了解新方法。通过尽早开放对新功能的讨论，并在实验版中提供它们，React 团队可以帮助库作者对新的 API 进行集成和测试，并帮助应用程序开发者着手适应新的思维方式和细微差别。

1.6　本书读者对象

本书适合希望学习 React 最新功能且具有一定经验的 JavaScript 开发者。本书侧重于 React Hooks、Concurrent 模式和 Suspense，并使用大量代码示例加快读者的理解，使读者能在自己的项目中充分使用这些功能(尽管并不是必须在生产中使用那些目前仍属于 React 实验版的功能)。除了提供简单实用的示例外，本书还深入探讨了一些功能背后的原理，以及开发者应该注意的细微差别。

总之，本书不是 React 的入门指南，因此不会详细介绍 React 的生态系统、构建工具、样

式和测试。读者应具备基本的 React 概念知识，并能够创建、构建和运行 React 应用程序。本书偶尔也会使用类组件的实例与新的函数式组件方法做比较，但不会重点介绍基于类的方法、高阶组件和渲染prop(不了解这些专业术语也不要担心，因为不必了解它们也可学习新概念)。

读者应该适应一些较新的 JavaScript 语法(如 const 和 let)，对象和数组解构，默认参数，扩展运算符，数组的方法(如 map、filter 和 reduce)。与类组件的比较，将会用到 JavaScript 的类语法，因此熟悉类语法是很有用的，但不是必须的。

1.7 开始吧

本书主要的代码示例为一个预订应用程序，可访问 GitHub 的 https://github.com/jrlarsen/react-hooks-in-action 网址下载，也可以在 Manning 网站的本书页面进行下载(www.manning.com/books/react-hooks-inaction)。示例应用程序开发中的每一步都位于一个单独的 Git 分支上，本书的代码清单包含了相关分支的名称。也可扫描封底二维码下载本书的代码清单。

1.8 本章小结

- 可使用 React 创建可复用的组件来组成应用程序，该组件可将状态转换为 UI。
- 可使用 JSX 和 prop 通过类似 HTML 的语法描述 UI。
- 可创建函数式组件，并搭配与组件相关的代码和功能。
- 可使用 React Hooks 为组件封装和共享功能，执行副作用，并 hook 到组件生命周期的各个阶段。
- 可创建自己的自定义 hook，并使用第三方库提供的自定义 hook。
- 可使用 Suspense 组件，为需要花费时间才能返回 UI 的组件提供回退方案。
- 可探索利用实验阶段的 Concurrent 模式处理内存中多个版本的 UI，从而在界面相应状态变化时，能够更轻松地从一个界面平滑过渡到另一个界面。
- 注意 React 的三个发布渠道：最新版、下一版和实验版。
- 可在 https://reactjs.org 上查阅 React 文档。