Web开发经典丛书

Redux in Action

Redux 实战

[美] 马克•加罗(Marc Garreau)
威尔•福罗(Will Faurot) 著
黄金胜 王冬阳 熊建刚 译

MANNING

清华大学出版社

Web 开发经典丛书

Redux 实战

[美] 马克·加罗(Marc Garreau)
威尔· 福罗(Will Faurot) 著
黄金胜 王冬阳 熊建刚 译

清华大学出版社
北 京

Marc Garreau, Will Faurot
Redux in Action
EISBN: 978-1-61729-497-6

图书在版编目(CIP)数据

Redux 实战 /(美)马克·加罗(Marc Garreau)，(美)威尔·福罗(Will Faurot) 著；黄金胜，王冬阳，熊建刚 译. —北京：清华大学出版社，2019
(Web 开发经典丛书)
书名原文：Redux in Action
ISBN 978-7-302-53033-6

Ⅰ. ①R… Ⅱ. ①马… ②威… ③黄… ④王… ⑤熊… Ⅲ. ①JAVA 语言－程序设计 Ⅳ. ①TP312.8

中国版本图书馆 CIP 数据核字(2019)第 094462 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：牛艳敏
责任印制：沈　露

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：三河市吉祥印务有限公司
经　　销：全国新华书店
开　　本：170mm×240mm　　印　　张：18　　字　　数：363 千字
版　　次：2019 年 7 月第 1 版　　印　　次：2019 年 7 月第 1 次印刷
定　　价：68.00 元

产品编号：081804-01

译 者 序

2015 年年中，我开始系统学习 React 与 Redux 技术栈。当时，前端技术领域的新技术层出不穷，React、Redux 等技术方兴未艾。

2012 年，我开始从后端开发转向前端开发工作，有幸见证了前端领域的发展、迭代与兴起。最令我感慨的是前端技术日新月异的发展速度，相信很多开发人员也有同感。因为我在工作中，学习过、也用到过很多前端技术。因此，亲身经历了很多前端技术的兴衰与淘汰。

由于前端技术的快速发展，前端开发人员也需要不断学习，而一些新技术刚出现时，学习材料仅有官方文档和一些博客文章，成体系又贴近实际开发工作的书籍资料相对缺乏。相信很多前端开发人员都有这样的经历：起初，踌躇满志、快速地学习了新技术的文档材料，但在公司实际的商业系统开发中，应用起来又困难重重，进展缓慢。

在最初学习 React 与 Redux 技术栈时，我也经历了一些困难和工作压力。彼时，我在国内某一线互联网公司工作。当时组内的前端技术栈以一套内部实现的 MVC 框架及相关衍生工具为主，框架已完成多年，经过几批不同维护者的迭代变更，已经变得陈旧、臃肿且难以学习和维护。所以，我最初学习 React 与 Redux 技术栈的目的之一，就是要在组内形成新技术的储备，并在新项目中试用积累。

然而，由于项目时间紧张且业务相对复杂，整个前端方向的推进遇到了很多困难。其中最大的困难，在于如何合理抽象相关业务，将它们与 Redux 很好结合起来。幸运的是，最终相关项目工作顺利完成。但在这个项目中经历的压力，也引发了我的一些思考：与传统的 MVC 型前端框架相比，Redux 和 React 技术栈在设计理念和实现上都有很大不同！只有深刻理解其精髓，才能不受经验束缚进而合理运用。

时至今日，使用 Redux 和 React 技术栈进行开发已经有几年时间了。期间不断学习思考，也不断在工作中实践，对 Redux 和 React 技术的理解自然也在加深。毫无疑问，Redux 这种前端数据(状态)管理技术的出现与流行，对前端开发领域具有重要意义，标志着前端开发更精细化，也更专业化。

回顾自己的学习历程，有经验也有教训。可以分享的经验是，新技术的学习要和实际的开发实践结合进行，尽可能做到学以致用，在实际中感受并验证所学，这样学

习效果和效用才能最大化。个人的重要教训是，在开始学习尝试一项新技术时，最好能找到合适的学习资料，相对系统全面地学习了解一遍相关的知识点，中间遇到暂时不能理解的内容也没有关系，知道后可以略过继续。这样做的好处是，能够快速建立对新技术知识的整体结构。了解全局的知识结构后，后续的学习实践很多都是对局部知识点认知的加深与强化。对于 Redux 初学者或者有一定经验的开发者，本书就是很好的学习材料。

近两年工作变得异常繁忙，但接到翻译本书的邀请后仍欣然接受。初衷之一也是希望能梳理自己的技术知识点，弥补自己之前学习认知的不足。参与本书翻译的还有我的两位同事：王冬阳和熊建刚。从工作实践角度讲，我们都参与了非常多的基于 React 和 Redux 技术栈的项目开发，有丰富的一线开发经验。但图书翻译这个领域，之前的相关经验有限，对此我们也诚惶诚恐，虽已努力追求更好的翻译效果，但纰漏在所难免，希望读者海涵，具体问题也可以联系沟通，共同讨论学习。

本书能顺利翻译并出版涉及很多人的认真工作，特别感谢清华大学出版社以及相关编辑老师！感谢对我们翻译工作的耐心支持，以及细心的译稿校对工作！希望通过本书，读者朋友能对 Redux 有更好的认知，在技术上取得更大进步！

黄金胜

序　言

Redux 是一款神奇的小工具。如你所见，它并没有涵盖太多的内容。在你喝完一杯咖啡之前，就可以熟悉它的每个方法。

Redux 不仅维护良好，而且还是成品。你可能经常听说 Redux 没有路线图，没有项目经理或看板。提交仍然被添加到 GitHub 存储库，但这些提交通常是对文档或官方示例的改进。

这怎么可能呢？你可能会发现将 Redux 视为架构模式很有帮助。你从 npm 安装的软件包是该模式的一个实现，它为你提供足够多的功能，使你的应用程序能够实现。

关键在于你能用这几个方法完成多少。事实上，Redux 模式可以完全解开一个 JavaScript 应用程序，留下更可预测、更直观和更具性能优势的内容。由于采用相同的模式，开发者工具还可以提供对应用程序状态和数据流前所未有的深入洞察。

但是有什么收获呢？所有软件选择都需要权衡，Redux 也不例外。成本是巨大的灵活性。这可能听起来像另一个优势，但它带来了有趣的挑战。Redux 模式并不由库或任何其他工具严格执行，并且不能希望这个小的软件包可以教育或指导开发者有效地使用此模式。

最后，开发人员可以找到自己的方式。这就解释了为什么 GitHub 存储库中的文档行数远远超过实现代码的行数。与官方文档一样出色的是，开发人员通常会从网络内外的分散资源中收集上下文和最佳实践：博客文章、书籍、推文、视频和在线课程等。

Redux 所允许的灵活性还带来了丰富的附加组件生态系统：选择器、增强器和中间件等。你将很难找到使用完全相同工具集的两个 Redux 应用程序。虽然每个项目都可以根据自己的独特需求定制工具，但对于新引入的开发人员来说，这可能是产生困惑的根源。Redux 新手经常发现，当他们不仅要掌握 Redux，而且还要考虑补充包的复杂性时，他们会沿着一条具有挑战性的学习曲线前进。这是我们想要撰写本书的主要原因：将我们的个人经验和知识从几十个不同的来源提炼成一个整洁、易用的包。

我们相信本书的真正价值在于能够指导你一步步地体验丰富的 Redux 生态系统。

这并不是对所有补充工具的全面看法。相反，我们选择了一些最常见的附加组件，这些附加组件可能会在其他资源中看到并且足够强大，足以应对任何客户端项目。请带上本书，快乐地阅读！我们很高兴你选择与我们共度美好时光。

作 者 简 介

Marc Garreau 是以太坊(Ethereum)基金会 Mist 核心团队的一名开发人员，他长期思考和研究 Mist 浏览器中的应用状态管理。在这之前，他在 Cognizant 和 Quick Left 咨询公司使用 Redux 设计和开发应用程序。他撰写了许多流行的 Redux 博客文章，并在丹佛地区的几个 JavaScript 会议上发表过演讲。

Will Faurot 是 Instacart 的一名全栈开发人员，他在 Instacart 从事多个面向用户产品的开发。他酷爱所有关于前端的内容，擅长用 React 和 Redux 构建复杂的用户界面。在过去的生活中，他教过网球，还录制过复古和蓝草音乐。如果你在海湾区域的一个安静夜晚仔细聆听，可能会听到他弹奏班卓琴的声音。

致　　谢

撰写一本书是一项艰巨的任务。有很多人对这个过程至关重要，无论是直接的还是间接的，将他们全部署名可能需要占用本书太多篇幅。本书之所以得以顺利出版，是因为站在巨人的肩膀上。

强大的社区是所有成功软件的基础。Redux 社区非常强大，我们感谢所有在博客中分享自己技能的人，在 GitHub 上帮助 Redux 用户处理问题的人，以及在全球众多 Redux 用户经常访问的在线平台上回答问题的人。

首先最重要的是，如果没有 Redux 的创作者 Dan Abramov 和 Andrew Clark，本书是不可能出版的。除了花时间研究和实现 Redux 外，他们在过去几年里花了无数小时来支持开发人员。我们还要感谢 Redux 的现有维护者 Mark Erikson 和 Tim Dorr。除了定期维护之外，例如，回答问题和合并代码，他们还自愿在几个不同的平台上花时间。这些人一起为本书做了大量的研究，没有他们就没有本书。无论是权衡最佳实践、编写文档，还是向好奇的开发人员提供反馈，他们的价值不可估量。我们感谢他们。

感谢 Manning 的整个团队，包括我们所有的编辑，感谢他们的指导和支持。特别要感谢 Ryan Burrows 提供的宝贵反馈，帮助我们改进了本书的代码，以及 Mark Erikson 花时间整理了一篇精彩的前言。我们还想感谢那些花时间阅读和评论本书的评论家：Alex Chittock、Clay Harris、Fabrizio Cucci、Ferit Topcu、Ian Lovell、Jeremy Lange、John Hooks、Jorge Ezequiel Bo、Jose San Leandro、Joyce Echessa、Matej Strasek、Matthew Heck、Maura Wilder、Michael Stevens、Pardo David、Rebecca Peltz、Ryan Burrows、Ryan Huber、Thomas Overby Hansen 和 Vasile Boris。感谢大家。

特别感谢我们的 MEAP 读者和论坛参与者，他们的反馈和鼓励对本书的顺利出版至关重要。

—Marc Garreau 和 Will Faurot

首先感谢我的妻子 Becky，她做出了很大牺牲：和写书的人一起生活。我保证我可能不会再写书了。感谢我的家人真实地反映了我内心的兴奋，即使我写的书是关于小毛虫的。感谢我的朋友鼓励我，帮助我战胜冒名顶替症，并给我提供健康的娱乐方式。更要感谢 Jeff Casimir、JorgeTéllez、Steve Kinney、Rachel Warbelow、Josh Cheek

和 Horace Williams，他们为 Will 和我在这个行业打开大门。感谢 Ingrid Alongi 和 Chris McAvoy 帮助我在职业生涯里塑造情感技术领导力。

—Marc Garreau

首先感谢我的父母，没有你们给予的指导、热情和鼓励，我就不会成功。你们帮助我意识到这样的事情是可能的。你们教我如何相信自己。谢谢。

感谢我的家人，感谢你们的支持与关爱。

感谢 Instacart 的每个人，你们通过提供反馈或讨论想法为我提供帮助。特别要感谢 Dominic Cocchiarella 和 Jon Hsieh。

最后，感谢 Lovisa Svallingson、Alan Smith、Allison Larson、Gray Gilmore、Tan Doan、Hilary Denton、Andrew Watkins、Krista Nelson 和 Regan Kuchan，你们在整个写作过程中提供了非常宝贵的反馈和鼓励。你们是我遇见的最好的朋友和软件知己。

—Will Faurot

前　言

自 2015 年年中发布以来，Redux 引起了 JavaScript 世界的关注。从它作为会议演示的概念验证和“只是另一个 Flux 实现”标签的简单开端，已经发展成为 React 应用程序中使用最广泛的状态管理解决方案。它也被 Angular、Ember 和 Vue 社区采用，并启发了许多模仿品和衍生产品。

我最喜欢引用的一句话是，“Redux 是一个通用框架，它提供了足够结构化和足够灵活性的平衡。因此，它为开发人员提供了一个平台，可以让他们为自己的用例构建自定义状态管理，同时能够重用图形化调试器或中间件之类的东西。”[1]的确，Redux 提供了一组基本的工具以供使用，并概述了组织应用程序更新逻辑的一般模式，最终由你来决定如何围绕 Redux 构建应用程序。你可以设计应用程序的文件结构，编写 reducer 逻辑，连接组件，并确定要在 Redux 上使用多少抽象。

Redux 的学习曲线有时会很陡峭。对于来自面向对象语言的大多数开发人员来说，函数式编程和不可变性是不熟悉的概念。编写另一个 TodoMVC 示例并没有真正展示 Redux 的好处，也不能解决构建“真实”应用程序的问题。但最终的收益是值得的。能够清楚地追踪应用程序中的数据流并了解特定状态变更的位置/时间/原因/方式是非常有价值的，并且良好的 Redux 使用方式最终会让代码在长期内更易于维护和可预测。

我大部分时间都是 Redux 维护人员，通过回答问题、改进文档和撰写教程博客来帮助人们学习 Redux。在这个过程中，我看过数百种不同的 Redux 教程。有鉴于此，我非常乐意推荐将《Redux 实战》一书作为学习 Redux 的最佳资源之一。

通过《Redux 实战》一书，Marc Garreau 和 Will Faurot 写了我希望自己写的 Redux 书籍。它非常全面、实用，并且可以很好地为开发现实世界中的 Redux 应用程序讲授许多关键的主题。我特别欣赏本书所涵盖的领域，并不总是有明确的答案，如构建项目，通过列出利弊，让读者知道这是一个他们可能不得不自己决定的领域。

在当今快速发展的编程世界中，没有一本书可以包罗有关工具的所有知识。但是，

1　来自 Facebook 工程师 Joseph Savona(https://github.com/reactjs/redux/issues/775#issuecomment-257923575)。

《Redux 实战》将为你打下坚实的基础和对 Redux 基础知识的理解——各部分如何结合，如何将这些知识用于实际应用程序，以及在何处寻找更多信息。我很高兴看到本书出版，并期待你加入 Redux 社区！

Mark Erikon
Redux 核心维护者

关 于 本 书

Redux 是一个状态管理库，旨在简化复杂用户界面的构建。它通常与 React 一同使用，在本书中亦如此，但它也在其他前端库(如 Angular)中日趋流行。

在 2015 年，React 生态系统亟待 Redux 的到来。在 Redux 之前，Flux 架构模式是一次令人兴奋的突破，世界各地的 React 开发人员都在尝试实现。数十个库受到空前的关注和使用。最后，兴奋变得疲惫。React 应用程序中的状态管理方式让人眼花缭乱。

Redux 发布后立即开始飞速发展，并很快成为最受推荐的受 Flux 启发的库。它使用单一数据源，着眼于不可变性，以及令人惊叹的开发体验，都证明 Redux 是一个更简单、更优雅、更直观的解决方案，解决了现有 Flux 库所面临的大多数问题。对于复杂应用程序中的状态管理，虽然仍有多个选择，但是对于那些喜欢类似 Flux 模式的开发人员来说，Redux 已经成为默认选择。

本书将带你了解 Redux 的基本原理，然后再继续探讨强大的开发者工具。我们将一起逐步开发一个任务管理应用程序，在该应用程序中，将探索真实世界中 Redux 的使用，并关注最佳实践。最后，将回到测试策略和构建应用程序的各种约定。

本书读者对象

读者应该熟悉 JavaScript(包括 ES2015)，并且至少掌握一些 React 的基础知识。不过，我们也了解到，许多开发人员都是在接触 React 的同时投身于 Redux。我们尽了最大努力来适应这类读者，我们相信他们通过一点额外的努力就可以阅读本书。虽然如此，我们的建议是在阅读这本书之前牢固地掌握 React 的基础知识。如果没有任何 React 开发经验，请考虑阅读《React 实践》和《快速上手 React 编程》。

本书的组织结构：路线图

本书共包含 12 章和 1 个附录。

第 1 章介绍 Redux 的诞生环境及其产生原因。你将了解 Redux 是什么，它的用途，

以及何时不应该使用它。该章总结了几种类似 Redux 的状态管理方法。

第 2 章直接开始开发第一个 React 和 Redux 应用程序。新功能的开发过程简单而快捷，如同旋风。你将从更高的角度了解其中每个角色：action、reducer 和 store 等。

第 3 章介绍强大的 Redux 开发者工具。Redux 开发者工具是 Redux 的最大亮点之一，本章将解释原因。

第 4 章最终将副作用引入从第 2 章开始的示例中。你将设置本地服务器，并在 Redux 模式中处理 API 请求。

第 5 章深入介绍一个更高级的特性：中间件。你将了解中间件在堆栈中的位置、它的功能以及如何构建自定义中间件。

第 6 章探讨用于处理更复杂的副作用的高级模式。你将学习如何使用 ES6 generator 函数，然后将学习如何使用 saga 管理长期运行的流程。

第 7 章重点介绍 Redux store 和视图之间的连接。你将了解选择器函数如何工作，然后使用 reselect 库实现一个健壮的解决方案。

第 8 章讨论一个常见问题——在 Redux store 中如何最好地构造数据。你将回顾到目前为止本书使用过的策略，然后探索另一种方法：范式化。

第 9 章回头讨论关于测试的所有内容。你将了解到一些流行的测试工具，如 jest 和 Enzyme，以及用于测试 Redux action、reducer 和 selector 等的策略。

第 10 章介绍如何保持应用程序精简和高效，涵盖了性能分析工具、React 最佳实践，以及用于提高性能的特定 Redux 策略。

第 11 章介绍组织 Redux 应用程序代码的几种策略。Redux 不介意内容的位置，所以你将学习已经成熟的常用惯例。

第 12 章提醒你，Redux 不仅仅可以管理 React Web 应用程序的状态。你将快速了解 Redux 可以在移动端、桌面端和其他 Web 应用程序环境中扮演的角色。

附录提供环境设置和工具安装的说明。

关于代码

大多数代码示例都是针对本书的示例应用程序Parsnip 的。这些示例以代码清单的形式呈现，其中多数都添加了注释，以保证代码清晰并说明某些代码选择背后的原因。正文中出现的代码以等宽字体显示，如 `like this`。

本书示例的源代码可从出版商网站 https://www.manning.com/books/redux-in-action 或 https://github.com/wfro/parsnip 下载。

一键安装脚本可用于 Mac OS X、Linux 和 Windows，它们与本书其他源代码放在一起。有关入门说明，请参阅附录。

软件需求

大多数代码示例，尤其是与示例应用程序相关的示例，都需要 Web 浏览器。我们推荐使用 Chrome，它完美支持 React 和 Redux 开发者工具。

我们用 create-react-app 构建示例应用程序，这不是严格要求，但强烈推荐。这是构建现代 React 开发环境最愉悦的方式。

本书使用以下 Create React App 和 Redux 版本：

- Redux：3.7.2。
- Create React App：1.0.17。

图书论坛

购买本书的读者能够免费访问 Manning Publications 运营的私人 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。要访问论坛，请访问 https://forums.manning.com/forums/redux-in-action。你还可以在 https://forums.manning.com/forums/about 上了解有关 Manning 论坛和行为规范的更多信息。

其他线上资源

Redux 社区在几个不同的平台上非常活跃。我们建议使用以下所有资源来了解更多信息，帮助巩固概念，并提出问题：

- Reactiflux 是讨论 React 和 Redux 的主要聊天室，位于 https://www.reactiveFlux.com/。
- Redux 文档中的术语表。尽管 Redux 有种种优点，但它确实需要相当数量的术语。特别是对于初学者而言，返回去查阅术语表是非常有用的。有关详细信息，请参阅 https://github.com/reactjs/redux/blob/master/docs/ Glossary.md。
- Redux 官方文档位于 https://redux.js.org/。
- Mark Erikson 的“实用 Redux”系列博客。对于中高级 Redux 用户来说，这是更好的选择，因为它提供了更多对Redux 使用的深入探索。有关详细信息，请参阅 http://blog.isquaredsoftware.com/2016/10/practical-redux-part-0-introduction/。

目　录

第 1 章

Redux 介绍

本章涵盖：

- 定义 Redux
- 了解 Flux 与 Redux 之间的差异
- 使用 Redux 和 React
- 介绍 action、reducer 和 store
- 学习何时使用 Redux

在 2018 年，如果你进入任何一个 React Web 应用程序，很有可能就会发现是 Redux 在管理其状态(state)。然而，能够如此之快地达到这个地步，是非常了不起的。几年前，Redux 还尚未创建，而同期 React 却拥有蓬勃发展的活跃用户群。React 的早期使用者认为，他们已经找到视图层(view layer)——MVC(Model-View-Controller)前端框架拼图中的“V”——的最佳解决方案。尚未能达成共识的是，一旦应用程序的大小和复杂度达到现实中所需的规模，应该如何管理它们的状态。最终，Redux 解决了这一争论。

在本书的整个讲解过程中，将会以一个 React 应用程序为“镜头”，来探索 Redux 及其生态系统。正如你将了解到的，可以将 Redux 插入各种风格的 JavaScript 应用程序中，但是由于一些原因，React 是理想的练习场景。最主要的原因是，Redux 是在 React 的背景下创建的。你最有可能在 React 应用程序中遇到 Redux，而 React 并不知

道如何管理应用程序的数据层。下面不再赘述，让我们开始吧！

1.1 什么是状态

React 组件具有本地状态(Local State，又称组件状态)的概念。例如，在任何指定的组件中，可以用于跟踪输入字段的值，或者跟踪按钮是否被单击过。本地状态可以轻松管理单个组件的行为。然而，如今的单页面应用程序，通常需要同步复杂的状态网。层级嵌套的组件，可以基于用户已经访问过的页面、AJAX 请求的状态或用户是否已登录来呈现不同的用户体验。

让我们考虑一个涉及用户身份验证状态的用例。产品经理告诉你，当用户登录到一个在线商店时，导航栏应该显示用户的头像，该在线商店应首先展示离用户地址最近的商品，并且应该隐藏新闻邮件注册表单。在普通的 React 架构中，能做的仅限于在每个组件之间同步状态。最后，很可能会将用户身份验证状态和其他用户数据从一个顶级组件向下传递到每个嵌套组件。

这种架构存在几个缺点。在该过程中，数据可能在不需要它的组件间过滤，而非将其传递给组件的子级。在大型应用程序中，这可能会导致不相关的组件之间出现大量的数据移动，通过 props 向下传，或通过回调向上传。很可能，位于应用程序顶层的少数组件，最终会知道整个应用程序中用到的大多数状态。当达到一定规模后，维护和测试代码会变得难以为继。由于 React 并非旨在像其他 MVC 框架一样解决同样广泛的问题，因此，就存在弥合这些差别的机会。

考虑到 React，Facebook 最终推出了 Flux，这是一种用于 Web 应用程序的架构模式。Flux 在前端开发领域变得极具影响力，并开始转变人们对客户端应用程序中状态管理的看法。对于这种模式，Facebook 提供了自己的实现，但很快就出现十多个受 Flux 启发的状态管理库，争夺着 React 开发人员的注意力。

对于希望扩展应用程序的 React 开发人员，这是个混乱的时期。尽管已经看到 Flux 的亮点，但也应继续尝试，以寻找更优雅的方式来管理应用程序中复杂的状态。有段时间，新人遭遇了选择悖论，分裂的社区力量产生了如此多的选项，这诱发了焦虑。不过，令人惊讶和欣喜的是，一切已尘埃落定，Redux 成为其中明显的胜出者。

Redux 通过一个简单的前提，一份巨大的回报，以及一次令人难忘的介绍，彻底征服了 React 世界。这个前提是，使用纯函数将整个应用程序的状态存储在一个单独的对象中。这份回报则是，应用程序的状态将会是完全可预见的。对大多数早期用户而言，有关 Redux 的介绍来自于丹·阿布拉莫夫(Dan Abramov)在 2015 年 React 欧洲会议上的演讲，标题为 *Live React：Hot Reloading with Time Travel* (演讲视频可参考 https://www.youtube.com/watch?v=xsSnOQynTHs)。Dan 展示了打破已有工作流程的 Redux 开发经验，令与会者赞叹不已。一种称为热加载(hot loading)的技术，可在保持

已有状态的同时让应用程序实时更新，并且新的 Redux 开发者工具能够在应用程序状态间进行时间旅行——只需要单击一下，就能倒回并重放用户操作。这种综合效果为开发人员提供了强大的调试能力，在第 3 章中将会详细介绍。

为了理解 Redux，首先要适当介绍一下 Flux，它是由 Jing Chen 为 Facebook 开发的一种架构模式(关于 Jing Chen，详见 https://facebook.github.io/flux/docs/ videos.html)。Redux 及其许多替代方案都是 Flux 架构的变体。

1.2　什么是 Flux

最重要的一点在于 Flux 是一种架构模式。它是作为主流的 JavaScript MVC 模式的替代品被开发的，JavaScript MVC 模式因现存框架(例如 Backbone、Angular 或 Ember)的流行而普及。尽管每个框架对 MVC 模式都有自己的实现，但许多框架都有类似的问题：通常，模型(model)、视图(view)和控制器(controller)之间的数据流可能会难以理解追寻。

许多这类框架使用双向数据绑定(two-way data binding)，其中视图的改变会更新相应的模型，并且模型中的更改会更新相应的视图。当任何给定的视图能够更新一个或多个模型，而这个或这些模型又可以更新更多的视图时，就会无法很好把握预期的结果，某种程度上这也无可指责。Chen 争辩说，尽管 MVC 框架适用于较小的应用程序，但这些框架所采用的双向数据绑定模型的扩展性，不足以满足 Facebook 的应用程序规模。由于害怕杂乱的依赖网会产生意想不到的后果，Facebook 的开发人员对做出的更改变得担心。

Flux试图解决状态的不可预知性，以及模型与视图紧密耦合的架构的脆弱性。Chen 废弃了双向数据绑定模型，转而采用单向数据流。Flux 要求对状态的所有修改遵循单一的路径，而不允许每个视图(view)与对应的模型进行交互。例如，当用户单击表单中的 Submit 按钮时，会向应用程序的唯一 dispatcher(调度程序)发送一个操作(action)。然后，该 dispatcher 将数据发送到合适的数据存储进行更新。一旦更新后，视图将会知道要渲染的新数据。图 1.1 对这种单向数据流进行了说明。

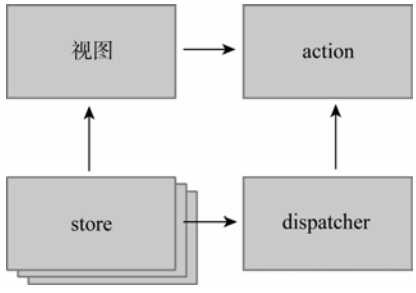

图 1.1　Flux 指定数据必须单向流动

1.2.1 action

状态的每一次改变都始于一个 action(参见图 1.1)，action 是描述应用程序事件的 JavaScript 对象。它们通常由用户交互或服务器事件(例如 HTTP 响应)产生。

1.2.2 dispatcher

Flux 应用程序中的所有数据流都通过单一的 dispatcher 进行汇集。dispatcher 自身的功能很少，因为其目的是接收所有的 action，并将它们发送到已注册的每个 store。每个 action 会被送到每个 store。

1.2.3 store

每个 store 管理应用程序中一个域的状态。例如，在电子商务网站中，可能会找到一个购物车 store 和一个产品 store。一旦把一个 store 注册到 dispatcher，它就开始接收 action。当 store 接收到它关心的 action 类型时，它就会进行相应的更新。一旦 store 产生更改，就会广播一个事件，让视图使用新的状态进行更新。

1.2.4 视图

Flux 可能在设计时考虑了React，但视图并不需要 React 组件。对视图而言，它们只需要订阅要显示的数据的 store。Flux 文档鼓励使用“控制器-视图”模式，借助一个顶级组件处理与 store 的通信，并将数据传递给子组件。让一个父组件和一个嵌套的子组件通过 store 通信，可能导致额外的渲染以及意想不到的副作用。

重申，Flux 首先是一种架构模式。Facebook 团队维护了这种模式的一个简单的实现方案，恰当地(或令人困惑地，取决于如何看待)命名为 Flux。自 2014 年以来，已出现许多替代实现方案，包括 Alt、Reflux 和 Redux。有关这些替代实现方案的更全面的列表请参见 1.6 节。

1.3 什么是 Redux

最好的解答来自官方文档：“Redux 是 JavaScript 应用程序的可预测状态的容器”(https://redux.js.org/)。Redux 是一个独立的库，但最常用作 React 的状态管理层。与 Flux 一样，其主要目标是为应用程序中的数据带来一致性和可预测性。Redux 将状态管理职责划分成以下几个独立的单元：

- store 将应用程序的所有状态都存储在单个对象中(通常将此对象称为对象树)。
- 只能使用 action 更新 store，action 是描述事件的对象。

- 被称作 reducer 的函数指定了如何转换应用程序的状态。reducer 是函数，它接收 store 中的当前状态和一个 action，然后返回更新后的下一个状态。

从技术上讲，Redux 可能不符合 Flux 的实现。它与 Flux 架构规定的若干构件有明显偏离，例如，完全移除了 dispatcher。最终，Redux 是类 Flux 的架构，而区别在于语义问题。

Redux 得益于源自 Flux 架构的可预测数据流，但它也找到了方法以降低 store 回调注册的不确定性。正如前面提到的，要调和多个 Flux store 的状态可能会很痛苦。相反，Redux 规定了一个单一的 store 来管理整个应用程序的状态。关于 Redux 的工作原理和相关的更多内容，后续章节将会有更多的介绍。

1.3.1　React 和 Redux

尽管 Redux 是基于 React 设计并开发的，但这两个库是完全解耦的。如图 1.2 所示，React 和 Redux 使用绑定进行连接。

图 1.2　Redux 不是任何现有框架或库的一部分，但称为绑定的附带工具将 Redux 和 React 连接了起来，本书将使用 react-redux 包

事实证明，Redux 的状态管理范式可以与大多数 JavaScript 框架一起实施。Angular、Backbone、Ember 及很多技术都支持绑定。

虽然本书基本上是关于 Redux 的，但也会将 Redux 与 React 紧密联系。尽管 Redux 是一个独立的小型库，但它与 React 组件非常吻合。Redux 有助于定义应用程序做什么，而 React 将处理应用程序的外观。

本书将会编写的大部分代码，以及阅读期间要编写的 React/Redux 代码，将分为以下几类：

- 应用程序的状态和行为，由 Redux 处理。
- 绑定——由 react-redux 包提供——用于连接 Redux store 中的数据和视图(React 组件)。
- 包含大部分视图层的无状态组件。

你将会发现，React 是 Redux 天然的生态系统。React 为 Redux 引入并管理更大的应用程序状态敞开大门，同时 React 具有在组件内部直接管理状态的机制。如果对其余的生态系统感兴趣，第 12 章将探讨 Redux 与其他几个 JavaScript 框架之间的关系。

1.3.2 3 个原则

Redux 中的状态由单一的可信数据源(single source of truth)表示，是只读的，并且只能用纯函数进行修改。领会了这些也就有了坚实的基础。

单一数据源

与 Flux 架构规定的各种域的 store 不同，Redux 在 store 内部的对象中管理整个应用程序的状态。使用单一的 store 具有重要的意义，可以在单个对象中表示整个应用程序状态的能力，简化了开发人员的体验。通过思考应用程序流程，预测新操作的结果以及调试任何给定的 action 产生的问题，都会非常容易。时间旅行调试(time-travel debugging)的潜力，或在应用程序状态快照中来回穿梭的能力，正是发明 Redux 的首要动力。

状态是只读的

与 Flux 一样，action 是应用程序状态发起改变的唯一方式。在不通过 action 进行通信的情况下，零散的 AJAX 调用不能引起状态的改变。Redux 与许多 Flux 实现的不同，在于 action 不会导致 store 中的数据突变。相反，每个 action 都会产生一个崭新的状态实例来替换当前状态。

改变由纯函数进行

通过 reducer 接收 action。重要的是，这些 reducer 是纯函数。纯函数是确定的，在给定相同输入的情况下，它们总是产生同样的结果，并且过程中不会改变任何数据。如果 reducer 在生成新状态的同时改变现有的状态，则最终可能出现错误的新状态，也会丢失每个新 action 应该提供的可预测的事务日志。Redux 开发者工具，以及诸如撤销和重做等其他特性，依赖于由纯函数计算所得的应用程序状态。

1.3.2 工作流

前面已经简要提到过 action、reducer 及 store 等主题，但本节将对每个主题做更深入的介绍。在此重点介绍每个元素扮演的角色，以及它们如何协同工作以产生期望的结果。现在，不必担心更精细的实现详情，因为后续章节中会有足够的时间来运用将要探索的概念。

现代的 Web 应用程序最终是与处理事件相关的。事件可以由用户发起，例如导航到新页面或提交表单。也可以由另外的外部源发起，例如服务器响应。响应事件通常涉及更新状态并使用更新后的状态重新渲染。应用程序处理事件越多，需要追踪并更新的状态就越多。将这种情况和大部分这些事件都是异步发生的事实相结合，突然就

会遭遇维护大规模应用程序的真正障碍。

Redux 的职责是，创建应用程序中有关如何处理事件以及管理状态的结构。希望在此过程中人们更加高效、快乐。

来看看如何使用 Redux 和 React 在应用程序中处理单个事件。假设任务是实现社交网络的核心功能之一——添加一个帖子到活动流(activity feed)。图 1.3 展示了用户资料页面的一个快速模型，灵感可能是也可能不是源自 Twitter。

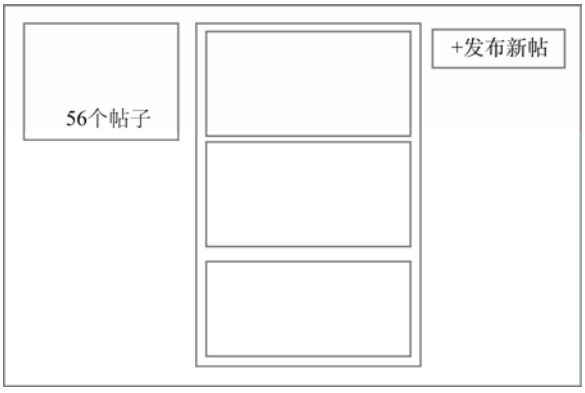

图 1.3　用户资料页面的一个快速模型。该页面由两个主要的数据支持：帖子总数和用户活动流中的帖子对象列表

处理诸如发布新帖之类的事件涉及以下不同步骤：

(1) 从视图中，指示事件已发生(帖子提交)并传递必要的数据(要创建的帖子的内容)。

(2) 根据事件的类型更新状态，在用户活动流中添加一项并增加帖子数。

(3) 重新渲染视图以反映更新后的状态。

听起来很合理吧？如果以前使用过 React，可能就直接在组件中实现了与此类似的功能。Redux 采用不同的方法，用于完成这 3 项任务的代码从 React 组件被移到了几个单独的实体中。尽管已经熟悉图 1.4 中的视图，但我们还是会兴奋地引入一组新的角色，希望你能够学习掌握。

图 1.4　看看数据如何在 React/Redux 应用程序中流动，我们省略了一些常见部分，如中间件和选择器，这些内容将在后续章节中深入介绍

action

为响应用户提交新帖要做两件事：将帖子添加到用户活动流并增加总的帖子数。在用户提交之后，将通过派发一个 action 来启动该过程。action 是表示应用程序中事

件的 JavaScript 对象字面量，如下所示：

```
{
  type: 'CREATE_POST',
  payload: {
    body: 'All that is gold does not glitter'
  }
}
```

拆解来看，action 是一个具有以下两个属性的对象：

- type——表示正在执行的 action 类别的字符串。根据惯例，type 属性的值大写，并使用下划线作为分隔符。
- payload——提供执行 action 所需数据的一个对象。本例中，只需要一个字段：想要发布的消息的内容。名称“payload”(有效载荷)仅是一种流行的惯例。

action 具有作为审计的优势，它保留了应用程序中发生的所有事件的历史记录，包括完成事务所需的任何数据。在保持对复杂应用程序的掌控方面，其价值不可低估。一旦习惯使用具有高度可读性的流来实时描述应用程序的行为，就会发现没有了它便很困难。

在整本书中，将会经常回顾这个理念。可以将 Redux 视为对应用程序中发生了什么以及如何响应事件的解耦。action 处理发生了什么。action 描述一个事件，它不知道也不关心下游会发生什么。最终，沿途的某处需要指定如何处理 action。这听上去像是适合 reducer 完成的任务！

reducer

reducer 是负责更新状态以响应 action 的函数。它们是简单的函数，接收当前状态和一个 action 作为参数，并返回下一个状态，如图 1.5 所示。

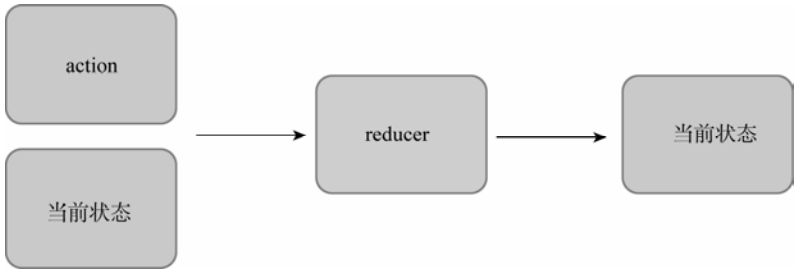

图 1.5 reducer 函数签名的抽象表示。这个图解看起来很简单，那是因为它本来就简单。reducer 是用于计算结果的简单函数，它们易于使用和测试

reducer 通常易于使用。与所有的纯函数类似，它们不会产生副作用。它们不会以任何方式影响外部，并且是引用透明的。相同的输入将始终产生相同的返回值。这使它们特别容易测试，给定特定输入，就可以验证是否会收到预期的结果。图 1.6 显示

了 reducer 如何更新帖子列表和总帖子数。

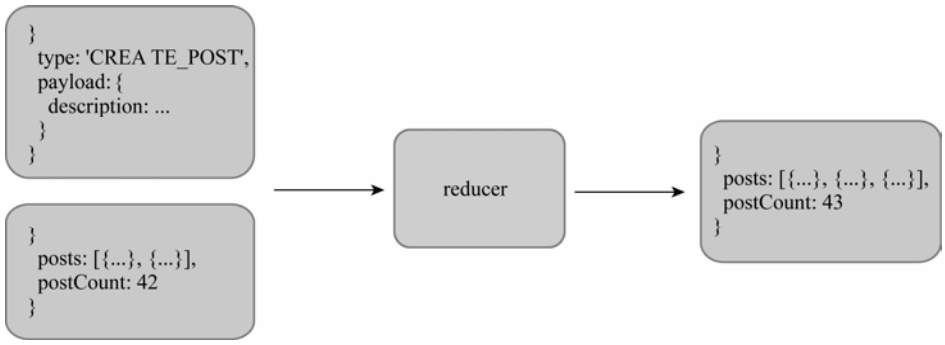

图 1.6　努力想象一个工作中的 reducer。它接收一个 action 和当前状态作为输入。reducer 的唯一职责是根据这些参数计算下一个状态。没有突变，没有副作用，没有业务逻辑。输入数据，输出数据

这个例子中关注的是单个事件，这意味着只需要一个 reducer。然而，肯定不仅限于(使用)一个。实际上，更大的应用程序通常实现若干个 reducer，每一个都会关注状态树的不同切片。这些 reducer 会被联合或组合成“根 reducer”(root reducer)。

store

reducer 描述了如何更新状态以响应 action，但它们无法直接修改状态。这种特权仅限于 store 拥有。

在 Redux 中，应用程序状态存储在单个对象中。store 拥有以下几个主要角色，其中包括：

- 持有应用程序状态。
- 提供一种访问状态的方式。
- 提供一种方式来指定对状态的更新。store 要求派发一个 action 来修改状态。
- 允许其他实体订阅更新(React 组件属于这种情况)。通过 react-redux 提供的视图绑定可以接收来自 store 的更新，并在组件中对其做出响应。

reducer 处理 action 并计算下一个状态。然后轮到 store 更新自身，并将新状态广播给所有已注册的监听者(请特别关注组成个人资料页面的组件)，参见图 1.7。

熟悉了最重要的几个构件后，我们来看一张更全面的 Redux 架构图。虽然其中几块现在还不熟悉，但在本书中将会反复提到此图(参见图 1.8)，而随着时间的推移，每个空白都会被填补。

回顾此图，与视图的交互可能会引发一个 action。该 action 将通过一个或多个 reducer 进行过滤，并在 store 中生成新的状态树。一旦状态更新，视图将意识到有新的数据要渲染。这就是整个循环！图 1.8 中带有虚线边框的条目(action创建器、中间件和选择器)是 Redux 架构中可选但又强大的工具。后续章节中会介绍这些主题。

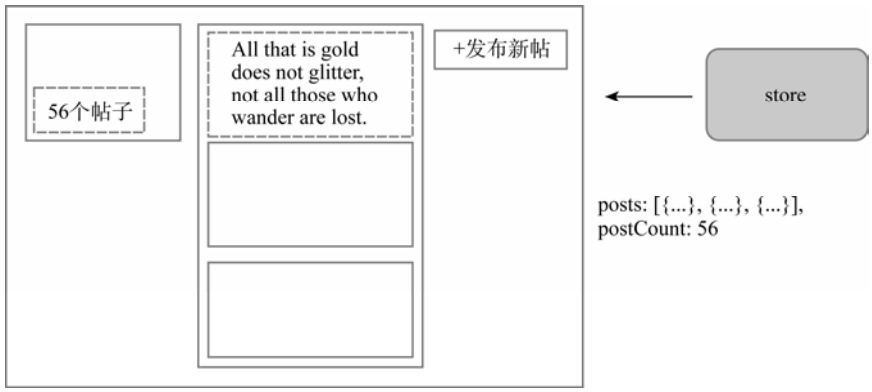

图 1.7 通过向个人资料页面提供新的状态，store 完成了循环。请注意，帖子数已增加，并且新帖子已被添加到活动流。如果用户想要添加其他帖子，将遵循与此相同的流程。视图派发 action，reducer 指定如何更新状态，store 会把新状态广播返回给视图

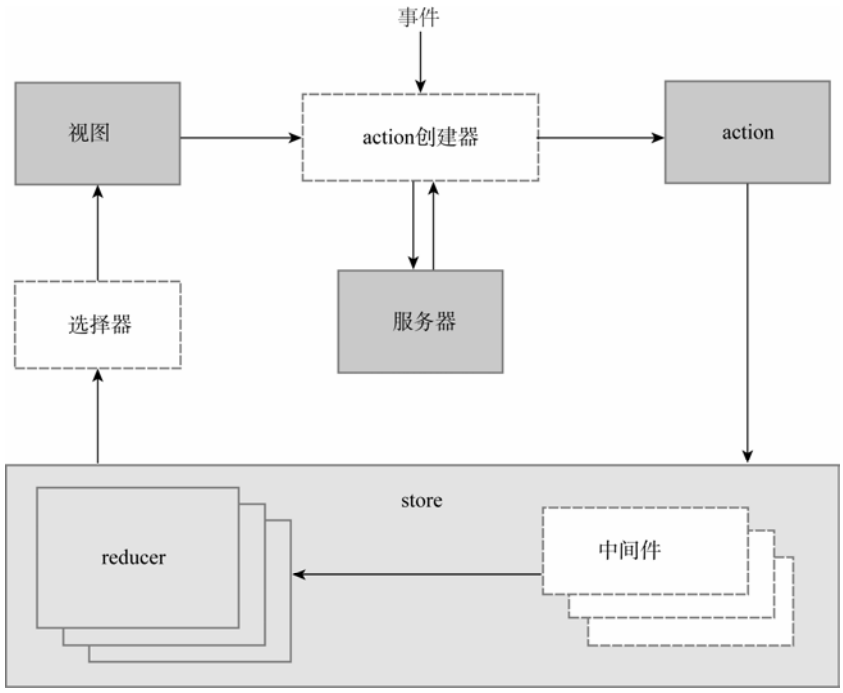

图 1.8 当继续前进时，此图将锚定你对 Redux 元素的理解。目前，已经讨论了 action、reducer、store 和视图

如果感觉这些内容太多了，请不要烦恼。正要探索的这种单向架构，对于新手，起初可能会不知所措(我们最初也这么认为)。领会这些概念需要一些时间。它们扮演什么角色，以及什么类型的代码属于哪里，要对此养成意识，既像是艺术也像是科学。

随着不断的练习，你将会掌握这种技能。

1.4　为什么要用 Redux

到目前为止，你已经触及 Redux 的许多要点。在完成第 1 章时，如果必须向老板推销 Redux，那么可以将这些理念整合到一个精彩的片段中。简而言之，Redux 是一个小型且易于学习的状态管理库，可以生成高度可预测、可测试并可调试的应用程序。

1.4.1　可预测性

Redux 的最大卖点是，它为处理应用程序的复杂状态带来了条理性。Redux 架构提供了一种直观的方式来概念化并管理状态，一次一个 action。无论应用程序的大小如何，单向数据流内的 action 会让针对单一 store 的更改是可预测的。

1.4.2　开发者体验

可预测性使出现世界一流的调试工具成为可能。热加载(hot-loading)和时间旅行(time-travel)调试工具为开发人员提供了更快的开发周期，无论是构建新功能还是寻找 bug。老板喜欢开心的开发人员，但更喜欢开发效率高的开发人员。

1.4.3　可测试性

需要编写的 Redux 实现代码主要是函数，其中许多都是纯函数。每一块都可以轻松分解并隔离进行单元测试。官方文档使用 Jest 和 Enzyme，但无论喜欢哪个 JavaScript 测试库，都可以完成测试。

1.4.4　学习曲线

Redux 是原生 React 的自然进阶，体积非常小，只公开了少量 API 用于完成工作。一天之内就可以熟悉 Redux 的所有内容。编写 Redux 代码还需要团队熟悉几种函数式编程范式。对于某些开发人员来说，这将是一个新的领域，但相关概念是清晰易懂的。一旦理解对于状态的更改只能由纯函数产生，就已经掌握了大部分内容。

1.4.5　体积

如果老板正在工作，那么检查清单中的重要一项就是项目依赖的大小。Redux 是个微型库——压缩后小于 7KB。胜出！

1.5 何时应该使用 Redux

虽然关于 Redux 如何出色已经介绍了很多，但 Redux 并非万能灵药。虽然讨论了为什么应该使用 Redux，但众所周知，天下没有免费的午餐，也不存在无须权衡折中的软件模式。

使用 Redux 的代价是会产生很多的模板代码，以及相比 React 本地组件状态更高的复杂度。重要的是要意识到：对于团队中的新开发人员来说，在他们能做出贡献之前，还需要学习 Redux 及其使用模式。

Redux 的联合创始人 Dan Abramov 在这一点上也有权衡考虑，甚至发表了一篇名为“你可能不需要 Redux”(*You Might Not Need Redux*)的博文(博客地址为 https://medium.com/@dan_abramov/you-might-not-need-redux-be46360cf367)。他建议开始时不使用 Redux，仅在遭遇足够多的状态管理痛点，能证明包含 Redux 是合理的之后才引入 Redux。该建议刻意描述得模糊不清，因为每个团队的转折点会略有不同。没有复杂数据需求的较小应用程序是最常见的情况，这种情况下不使用 Redux 而使用普通的 React 可能会更加合适。

那些痛点可能是什么样的呢？团队可以使用一些常见的场景来证明引入 Redux 的合理性。第一种场景是，在几层不会使用任何数据的组件间传递数据。第二种场景是，处理在应用程序的不相关的部分之间共享和同步数据。对于使用 React 实现这些任务，我们都有一定容忍度，但最终会有容忍极限。

如果知道自己想要构建的特定功能是 Redux 擅长的，Redux 很可能会是不错的选择。如果知道应用程序会有复杂的状态，并且需要撤销与重做功能，请直截了当地引入 Redux。如果需要服务端渲染，请优先考虑 Redux。

1.6 Redux 的备选方案

如前所述，Redux 进入了拥挤的状态管理市场，而此后也出现了更多的可选方案。来看一下用于 React 应用程序状态管理的最流行的替代方案。

1.6.1 Flux 的一些实现

在研究时，我们数了数，有 20 多个 Flux 实现的库。令人惊讶的是，至少其中有 8 个已经在 GitHub 上收到超过 1000 个星星。这突显了 React 历史上的一个重要时代——Flux 是一种开创性理念，激发了社区的活力，并因此推动大量的试验与发展。在此期间，代码库以令人应接不暇的速度来来去去，这也造就了“JavaScript 疲劳”(JavaScript Fatigue)一词。事后来看，很明显，这些试验中的每一个都是途中重要的垫脚石。随着

时间的推移，许多备选的 Flux 实现的维护者优雅地退出了比赛，转而支持 Redux 或其他一些流行的方案中的某个，但是仍然有几个维护良好的备选方案。

Flux

Flux 当然是这一切的开始。用维护者自己的话说，“Flux 更像一种模式而非框架”。在这个代码库中，能找到大量有关 Flux 架构模式的文档，但对于方便地使用 Flux 构建应用程序，只暴露了少量的 API。dispatcher 是 Flux 的核心，事实上，其他几个 Flux 实现已将 dispatcher 整合到它们的库里。按 GitHub 上的星星个数来衡量，Flux 的流行度大概是 Redux 的一半，而且继续由 Facebook 团队积极维护。

Reflux

Reflux 是原始的 Flux 的快速跟随者。Reflux 将函数响应式编程(functional reactive programming)思想引入 Flux 架构中，剥离了单个的 dispatcher，进而让每个 action 都具有派发自身的能力。回调函数可以和更新 store 的 action 一起注册。Reflux 仍在维护，并且以 GitHub 上的星星个数衡量，它的流行度大概有 Redux 的六分之一。

Alt

与 Reflux 不同，Alt 符合原始的 Flux 理念并使用 Flux dispatcher。Alt 的卖点在于它对 Flux 架构的坚持并减少了模板代码。Alt 曾经拥有一个活跃的社区，但在撰写本书时，已经有 6 个月没有提交记录了。

荣誉榜单

在 GitHub 上超过 1000 个星星的(Flux 实现)库中，还有 Fluxible、Fluxxor、NuclearJS 以及 Flummox。Fluxible 继续由 Yahoo 团队维护。Fluxxor、NuclearJS 及 Flummox 可能还在维护，但不再活跃。强调这些项目是因为它们是重要的垫脚石，Flummox 是由 Andrew Clark 创建的，Andrew Clark 后来与 Dan Abramov 一起创建了 Redux。

1.6.2　MobX

MobX 为状态管理提供了一种函数响应式解决方案。与 Flux 类似，MobX 使用 action 修改状态，但组件会对变化后的或可观察的状态做出响应。虽然函数响应式编程中的部分术语令人生畏，但在实践中这些概念易于理解和接受。MobX 需要的模板代码相比 Redux 更少，但在底层做了很多事情，因此更含蓄。在 2015 年年初，MobX 的第一次提交仅比 Redux 早几个月。

1.6.3 GraphQL 客户端

GraphQL 是一项激动人心的新技术，也是由 Facebook 团队开发的。它是一种查询语言，允许准确指定并接收组件所需的数据，非常适合 React 组件预期的模块化，组件所需的任何数据的获取都封装在其中。API 的查询针对父子组件的数据需求进行了优化。

通常，GraphQL 与 GraphQL 客户端一起使用。如今最受欢迎的两个客户端是 Relay 和 Apollo。Relay 是由 Facebook 团队(和开源社区)开发维护的另一个项目。起初，Apollo 的底层是使用 Redux 实现的，但现在提供了额外的可配置性。

虽然可以同时引入 Redux 和 GraphQL 客户端来管理同样的应用程序状态，但你会发现该组合可能过于复杂且不必要。虽然 GraphQL 处理数据从服务器的获取，而 Redux 更通用一些，但它们的用途有重叠之处。

1.7 本章小结

本章介绍了 Flux 架构模式以及 Redux 的运行，你了解了有关 Redux 的一些实际细节。

现在，你已经准备好将基本的构件放在一起，并端到端地查看运行中的 Redux 应用程序。在下一章中，将使用 Redux 和 React 构建一个任务管理应用程序。

本章的学习要点如下：

- Redux 的状态存储在单个对象中，并且是纯函数的产物。
- Redux 可以引入可预测性、可测试性以及复杂应用程序的可调试性。
- 如果经历过在应用程序中同步状态或在多层级组件间传递数据的痛点，请考虑引入 Redux。

第 2 章

第一个 Redux 应用程序

本章涵盖：

- 配置 Redux store
- 使用 react-redux 连接 Redux 与 React
- 使用 action 和 action 创建器(action creator)
- 使用 reducer 更新 state
- 理解容器和展示型 React 组件

现在，你一定迫不及待地想要开始编写一个 Redux 应用程序了。既然已经掌握了足够的背景知识，那就让我们开始吧！本章将指导你搭建和开发一个简单的任务管理应用程序，并使用 Redux 来管理应用程序状态。

本章结束后，你将经历一个完整应用程序的创建过程，但更为重要的是，你学到的基础知识将足够你独立开发简单的 Redux 应用程序。通过学习组件(我们特意没有把这部分内容放在第 1 章)，你能更好地理解单向数据流以及程序的每个地方是如何践行这一思想的。

你可能会认为在本章即将构建的小型应用程序中使用 Redux 有些大材小用了。重申一下第 1 章中提及的一个观点，除非能通过引入 Redux 解决遇到的诸多痛点，否则我们还是建议只使用 React。

如果本章内容已经是本书的全部，那么引入 Redux 确实是大材小用。随着后续章