第 1 章　为什么选择 React

正在阅读这本书的读者，可能已经对 React 有所了解。但如果您不熟悉 React，不用担心，本书已经将哲学定义降到最低。然而，本书涵盖了丰富的内容，所以我认为设定基调是第一步。我们的目标是学习 React 和 React Native，但也会构建一个可扩展和适应性强的架构，能够应对今天和未来使用 React 构建的一切内容。换句话说，我们想要围绕 React 打造一个基础框架，配备一套能够经得起时间考验的额外工具和方法。本书将指导读者使用路由、TypeScript 类型系统、测试等工具。

本章首先简要解释 React 存在的原因。然后将思考 React 的简洁性以及它如何处理 Web 开发者面临的许多典型性能问题。接下来将讨论 React 的声明式哲学以及 React 程序员可以预期的工作抽象层次。之后将触及 React 的一些主要功能。最后将探讨如何设置一个项目以开始使用 React。

一旦对 React 有了概念性的理解以及它如何解决 UI 开发问题，读者将有能力理解本书其余部分的内容。本章主要涉及以下内容。

- React 是什么？
- React 中的新功能。
- 搭建一个新的 React 项目。

1.1　React 是什么

React 是一个用于构建用户界面的 JavaScript 库。这是一个非常完美的定义，因为事实证明，大多数事件正是我们所需要的。这一描述的优点在于它省略了其他所有内容。React 不是一个庞大的框架，也不是一个全栈解决方案，不会处理从数据库到通过 WebSocket 连接进行实时更新的所有事情。实际上，我们可能不需要这些预先打包的解决方案。如果 React 不是一个框架，那么它到底是什么呢？

1.1.1　React 仅仅是视图层

React 通常被认为是应用程序中的视图层。应用程序通常被划分为不同的层次，如视图

层、逻辑层和数据层。在这种情况下，React 主要用于处理视图层，这包括根据数据和应用程序状态的变化渲染和更新用户界面。React 组件改变了用户所看到的内容。如图 1.1 展示了 React 在前端代码中的定位。

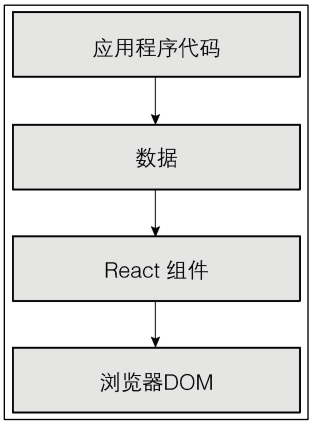

图 1.1　React 应用程序层

这就是 React 的全部核心概念。当然，当随着内容的逐步深入，这一主题会有微妙的变化，但流程或多或少是相同的。

（1）应用逻辑：从生成数据的一些应用逻辑开始。

（2）将数据渲染到用户界面：下一步是将这些数据渲染到用户界面。

（3）React 组件：为了完成这项任务，将数据传递给一个 React 组件。

（4）组件的作用：React 组件承担起将 HTML 渲染到页面上的责任。

React 看起来是另一种渲染技术。我们将在本章的剩余部分讨论 React 能够简化应用开发的一些关键领域。

1.1.2　化繁为简

React 没有太多需要学习和理解的复杂部件。虽然 React 拥有一个相对简单的 API，但要注意的是，在表面之下，React 以一定程度的复杂性运作。本书将深入探讨这些内部工作机制，探索 React 的架构和机制的各个方面，并为读者提供全面的了解。持有一个小的 API 工作的优势是，用户可以花更多的时间熟悉它，对其进行实验等。对于大型框架来说则恰恰相反，用户所有的时间都用于了解一切是如何工作的。图 1.2 给出了在用 React 编程时需要考虑的 API 的大致概念。

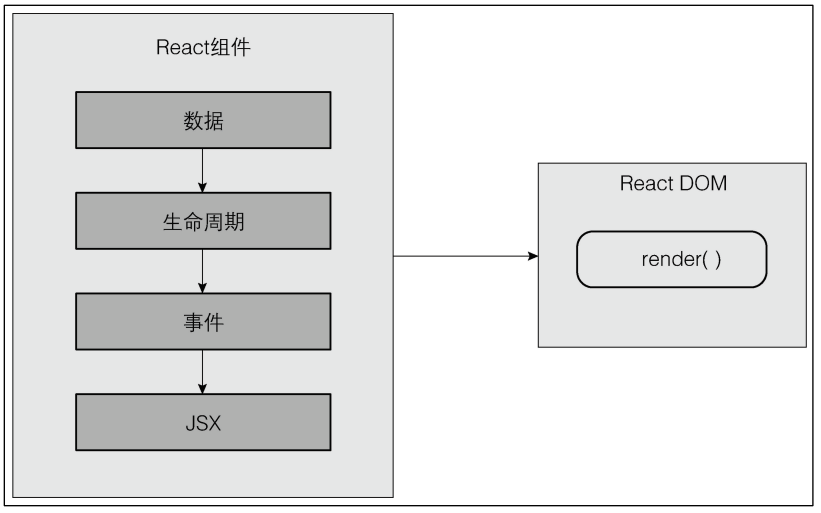

图 1.2　React API 的简洁性

React 分为两个主要的 API。

- React 组件 API：这些是由 React DOM 渲染的页面部分。
- React DOM：这是用于在网页上执行渲染的 API。

在 React 组件中，我们需要考虑以下几个方面。

- 数据：这是来自某处的数据（组件不关心来源），并由组件进行渲染。
- 生命周期：例如，生命周期的一个阶段是组件即将被渲染。在 React 组件中，方法或 Hooks 响应组件在 React 渲染过程中进入和退出阶段的变化，这些变化随着时间发生。
- 事件：我们编写的代码，用以响应用户交互。
- JSX：这是在 React 组件中描述 UI 结构常用的语法。尽管 JSX 与 React 密切相关，但它也可以与其他 JavaScript 框架和库一起使用。

不要过分纠结 React API 不同领域的具体含义。这里的要点是，React 本质上是简单的，这意味着不必在这里花费大量时间了解 API 细节。相反，一旦掌握了基础知识，我们可以花更多时间探讨与声明式 UI 结构完美融合的 React 使用模式。

1.1.3　声明式 UI 结构

React 新手很难理解组件如何将标记与 JavaScript 结合起来声明 UI 结构。如果用户在查看 React 示例时也有相同的不良反应，请不要担心。最初，我们可能会对这种方法持怀

疑态度，我认为原因在于，我们已经受到了关注点分离原则的熏陶。这一原则规定，逻辑和表现等不同的关注点应该相互独立。现在，每当看到事物被组合在一起时，我们就会本能地认为这是不好的，且不应该发生。

React 组件使用的语法称为 JSX（JavaScript XML 的简称，也称为 JavaScript 语法扩展）。一个组件通过返回一些 JSX 来渲染内容。JSX 本质上通常是 HTML 标记，加上 React 组件的自定义标签。基于此，本书将在接下来的章节中进行详细讲解。

关于声明式 JSX 方法的突破性之处在于，用户不必手动执行复杂的操作来改变组件的内容。相反，我们描述 UI 在不同状态下应该呈现的样子，React 会高效地更新实际的 DOM 以进行匹配。因此，React UI 变得更容易和更高效地工作，从而带来更好的性能。

例如，想使用 jQuery 来构建应用程序。有一个页面，上面有一些内容，当单击一个按钮时，在段落上添加一个类：

```
$(document).ready(function() {
  $('#my-button').click(function() {
    $('#my-paragraph').addClass('highlight');
  });
});
```

执行这些步骤已经足够简单。这被称为命令式编程，它在 UI 开发中存在一定的问题。命令式编程在 UI 开发中的问题在于，它可能导致难以维护和修改的代码。这是因为命令式代码通常紧密耦合，意味着对代码进行的一部分的更改可能会在其他地方产生意想不到的后果。此外，命令式代码难以推理，因为用户很难理解控制流程和应用程序在任何给定时间的状态。虽然更改元素类的示例很简单，但实际应用程序往往涉及 3、4 个步骤才能使某些事情发生。

React 组件不要求以命令式的方式执行相关步骤。这就是为什么 JSX 对 React 组件至关重要。XML 风格的语法使描述 UI 应该是什么样子变得容易。也就是说，组件将要渲染的 HTML 元素是什么？

```
export const App = () => {
  const [isHighlighted, setIsHighlighted] = useState(false);
  return (
    <div>
      <button onClick={() => setIsHighlighted(true)}>Add Class</button>
      <p className={isHighlighted && "highlight"}>This is paragraph</p>
    </div>
  );
};
```

在这个例子中，不仅仅是在编写浏览器应该执行的命令式过程。这更像是一个指令，说明 UI 应该是什么样子以及在它上面应该发生什么用户交互。这就是声明式编程，非常适合 UI 开发。

一旦声明了 UI 结构，随后就需要指定它是如何随时间变化的。

1.1.4　数据随时间变化

另一个 React 新手难以把握的概念是，JSX 就像一个静态字符串，代表一块渲染输出。这就是数据和时间流逝发挥作用的地方。React 组件依赖传入的数据。这些数据代表了 UI 的动态部分，例如，基于布尔值渲染的 UI 元素可能在组件下次渲染时发生变化。图 1.3 说明了这一理念。

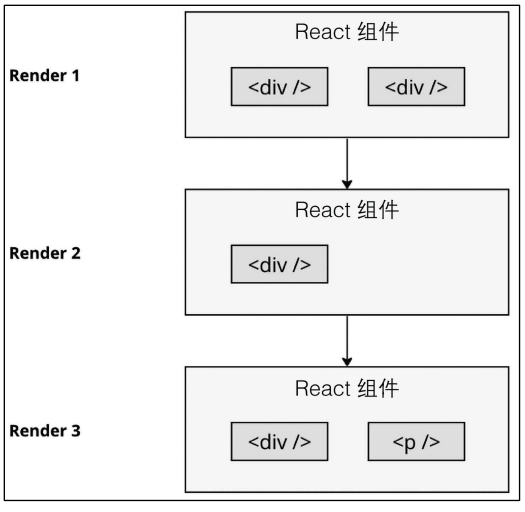

图 1.3　随时间变化的 React 组件

React 组件渲染时，就像在某个确切时刻对 JSX 拍摄快照一样。随着应用程序在时间轴上向前推进，用户拥有了一系列有序的渲染 UI 组件。除了声明式地描述 UI 应该是什么样，重新渲染相同的 JSX 内容也使开发者的工作变得更加容易。这里，挑战在于确保 React 能够处理这种方法所带来的性能需求。

1.1.5　性能至关重要

使用 React 构建 UI 意味着可以用 JSX 声明 UI 的结构。这比逐件组装 UI 的命令式方

法出错更少。然而，声明式方法确实在性能方面带来了挑战。

例如，对于初始渲染来说，拥有一个声明式的 UI 结构是很好的，因为页面上还没有任何内容。因此，React 渲染器可以查看在 JSX 中声明的结构，并在浏览器的 DOM 中进行渲染。

注意

文档对象模型（DOM）代表了浏览器渲染后的 HTML。DOM API 是 JavaScript 能够改变页面内容的方式。

这一概念在图 1.4 中进行了说明。

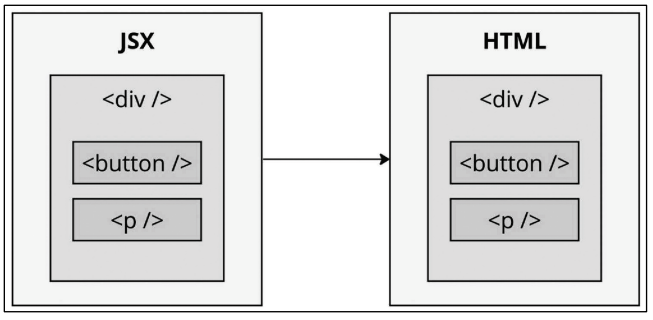

图 1.4　JSX 语法如何在浏览器 DOM 中转换为 HTML

在初始渲染时，React 组件及其 JSX 与其他模板库没有区别。例如，有一个名为 Handlebars 的模板库，用于服务器端渲染，它会将模板渲染为 HTML 标记字符串，然后将其插入浏览器 DOM 中。React 与 Handlebars 等库的不同之处在于，React 能够适应数据变化时需要重新渲染组件的情况，而 Handlebars 只会重建整个 HTML 字符串，就像在初始渲染时那样。由于这在性能上存在问题，最终通常会实施命令式的变通方法，手动更新 DOM 的小部分内容。我们最终得到的是声明式模板和命令式代码的混乱局面，用以处理 UI 的动态方面的内容。

我们在 React 中没有这样做。这就是 React 与其他视图库的不同之处。组件在初始渲染时是声明性的，即使在重新渲染时也是如此。正是 React 在后台所做的工作，使得重新渲染声明式 UI 结构成为可能。

在 React 中，当创建一个组件时，我们会清晰而直接地描述它应该呈现的样子。即使在更新组件时，React 也会在幕后平滑地处理变化。换句话说，组件在初始渲染时是声明式的，并且在重新渲染时也保持这种风格。这是因为 React 使用了虚拟 DOM，它用于在内存中保存真实 DOM 元素的表示。这样做的目的是，每次重新渲染一个组件时，它可以将新内容与页面上已经显示的内容进行比较。基于差异，虚拟 DOM 可以执行必要的命令式步

骤来进行更改。因此，我们不仅在需要更新 UI 时能够保留声明式代码，而且 React 还会确保这一过程以一种高效的方式完成。该过程如图 1.5 所示。

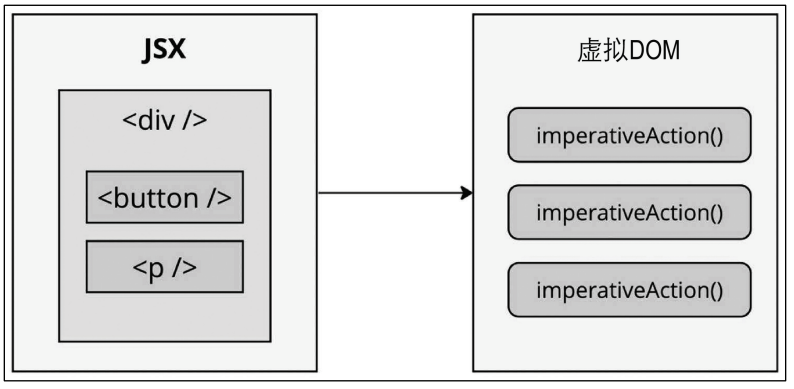

图 1.5　React 将 JSX 语法转译成命令式的 DOM API 调用

注意

当阅读有关 React 的资料时，经常会看到诸如 diffing（差异比较）和 patching（修补）之类的词汇。diffing 是指比较旧内容（UI 的先前状态）与新内容（更新后的状态）以识别差异，这很像比较两个版本的文档以查看变化。patching 是指执行必要的 DOM 操作来渲染新内容，确保只进行特定的更改，这对于性能至关重要。

与任何其他 JavaScript 库一样，React 受限于主线程的运行至完成（run-to-completion）的特性。例如，如果 React 虚拟 DOM 逻辑正忙于比较内容和修补真实的 DOM，浏览器就无法响应用户的输入，如单击操作或交互行为。

稍后将会看到，React 对内部渲染算法进行了修改，以缓解这些性能问题。随着性能问题的解决，我们需要确保 React 足够灵活，从而能够适应未来可能想要部署应用程序的不同平台。

1.1.6　正确的抽象级别

在深入研究 React 代码之前，需要在高层次上介绍的另一个主题是抽象。

上一节讨论了 JSX 语法如何转换为更新用户界面的底层操作。要了解 React 如何转换声明式 UI 组件，一个更好的方法是，不必关心渲染目标是什么。React 的渲染目标是浏览器 DOM，但正如我们将要看到的，它并不局限于浏览器 DOM。

React 有可能被用于在任何设备上创建的任何用户界面。我们才刚刚开始通过

React Native 看到这一点，但其可能性是无穷无尽的。如果 React Toast（尚不存在）突然变得与我们息息相关，也不必感到惊讶。React 的抽象层实现了一种平衡，在保持实用、高效的用户界面开发方法的同时，实现了多功能性和适应性。

通过图 1.6，用户可以了解 React 的目标不仅仅是浏览器。

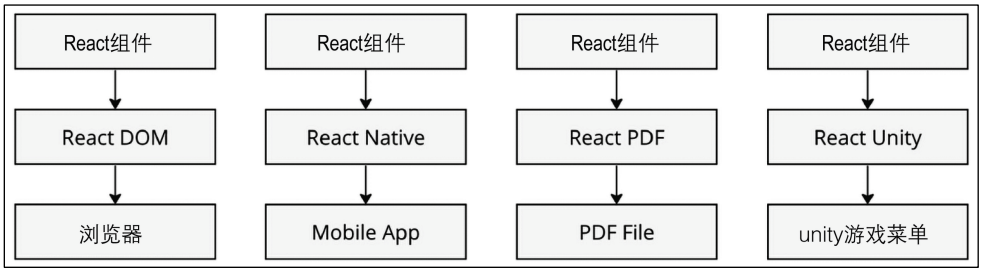

图 1.6　React 将目标渲染环境从实现的组件中抽象出来

从左到右依次是 React DOM、React Native、React PDF 和 React Unity。所有这些 React 渲染器库都接收 React 组件，并返回特定平台的结果。正如用户所看到的，要实现新的目标，同样的模式也适用以下场合。

- 针对目标实现特定的组件。
- 实现一个 React 渲染器，它可以在后台执行特定于平台的操作。

显然，这只是对任何特定 React 环境的实现内容的过度简化。但细节并不重要。重要的是，我们可以利用 React 知识，在任何平台上专注于描述用户界面的结构。

现在已经了解了抽象在 React 中的作用，让我们来看看 React 的新功能。

1.2　React的新功能

React 是一个在不断变化的 Web 开发领域持续进化的库。当读者开始学习并掌握 React 的进程时，了解该库的演变及其随时间的更新是很重要的。

React 的一个优势是其核心 API 在最近几年保持了相对稳定。这提供了一种连续性，允许开发者利用他们从早期版本中获得的知识。React 的概念基础保持不变，这意味着 3 年或 5 年前获得的技能今天仍然适用。React 从 React 0.x 到 React 18 版本，进行了许多关键的更改和增强，如下所示。

- React 0.14：这个版本引入了函数组件，允许开发者将函数用作组件，简化了基本 UI 元素的创建。当时，没有人知道现在我们将只编写功能组件，而几乎完全放弃基于类的组件。

- React 15：采用新的版本命名方案，React 15 的下一个更新带来了内部架构的彻底改革，从而提高了性能和稳定性。
- React 16：这个版本是 React 历史上最值得关注的发布之一。它引入了 Hooks，这是一个革命性的概念，使开发者能够在不需要类组件的情况下使用状态和其他 React 特性。Hooks 使代码更简单、更易读，改变了开发者编写组件的方式。在本书中，我们将探索许多 Hooks。此外，React 16 还引入了 Fiber，这是一种新的调和机制，特别是在处理动画和复杂的 UI 结构时，显著提高了性能。
- React 17：这个版本专注于更新和维护与以前版本的兼容性。它引入了一个新的 JSX 转换系统。
- React 18：此版本继续沿着改进的轨迹前进，并强调性能增强和新增功能，如自动批量渲染、状态转换、服务器组件和流式服务器端渲染。大部分与性能相关的重要更新将在第 12 章中探讨。关于服务器渲染的更多细节将在第 14 章中讨论。
- React 19：此版本引入了几个重大特性和改进。React 编译器是一个新编译器，它支持自动记忆化并优化重新渲染，消除了手动使用 useMemo、useCallback 和 memo 优化的需求。增强的 Hooks 简化了常见任务，如用于数据获取的 use（promise）、用于表单处理的 useFormStatus()和 useFormState()，以及用于 UI 的 useOptimistic()。React 19 还带来了简化的 API，如 ref 成为常规属性，React.lazy 被替换，Context. Provider 简化为 Context。异步渲染允许在渲染期间异步获取数据，而不会阻塞 UI，同时错误处理改进提供了更好的机制来诊断和修复应用程序中的问题。

React 的稳定性和兼容性使其成为一个适合长期使用的可靠库，而持续的更新确保它始终保持在 Web 和移动开发领域的前沿。在本书中，所有示例都将使用最新的 React API，确保它们在未来版本中保持功能性和相关性。

现在我们已经探讨了 React 的演变和更新，接下来可以更深入地研究 React，并检查如何设置新的 React 项目。

1.3　搭建一个新的 React 项目

开始时，有几种方式可以创建一个 React 项目。本节将探讨 3 种常见的方法。

- 使用 Web 打包工具。
- 使用框架。

- 使用在线代码编辑器。

注意

要开始开发和预览 React 应用程序，首先需要在计算机上安装 Node.js。Node.js 是一个用于执行 JavaScript 代码的运行时环境。

接下来将详细介绍每一种方案。

1.4　使用 Web 打包工具

使用 Web 打包工具是创建 React 项目的高效方式，特别是构建单页应用程序（SPA）时。本书的所有示例将使用 Vite 作为 Web 打包工具。Vite 以其卓越的速度和易于设置及使用而闻名。

要使用 Vite 设置项目，需要执行以下步骤。

（1）确保计算机上安装了 Node.js，方法是访问官方 Node.js 网站（https://nodejs.org/）并下载适合操作系统的版本。

（2）打开终端或命令提示符，并访问想要创建项目的目录。

```
mkdir react-projects
cd react-projects
```

（3）运行以下命令并使用 Vite 创建一个新的 React 项目。

```
npm create vite@latest my-react-app -- --template react
```

此命令将创建一个名为 my-react-app 的新目录，并使用 Vite 模板设置 React 项目。

（4）项目创建完成后，终端应如下所示。

```
Scaffolding project in react-projects/my-react-app...
Done. Now run:

  cd my-react-app
  npm install
     npm run dev
```

（5）进入项目目录并安装依赖项。终端中的结果显示应如下所示。

```
added 279 packages, and audited 280 packages in 21s
```

```
103 packages are looking for funding
  run 'npm fund' for details

found 0 vulnerabilities
```

最后，通过运行以下命令启动开发服务器：npm run dev

此命令启动开发服务器，用户可以通过打开浏览器并访问 http://localhost:3000 来查看 React 应用程序。

至此，我们已经成功使用 Vite 作为 Web 打包工具搭建了 React 项目。有关 Vite 及其可能的配置的更多信息，请访问官方网站 https://vitejs.dev/。

1.4.1　使用框架

对于现实世界和商业项目，建议使用在 React 之上构建的框架。这些框架开箱即提供额外的功能，如路由和数据资源管理（图像、SVG 文件、字体等）。它们还指导用户有效地组织项目结构，因为框架经常强制执行特定的文件组织规则。一些流行的 React 框架包括 Next.js、Gatsby 和 Remix。

在第 13 章中，我们将探索设置 Next.js 以及它与使用普通 Web 打包工具之间的一些差异。

1.4.2　在线代码编辑器

在线代码编辑器结合了 Web 打包工具和框架的优势，允许用户在云端或直接在浏览器内设置 React 开发环境。这消除了在机器上安装内容的需要，进而直接在浏览器中编写和探索 React 代码。

有各种在线代码编辑器可供选择，其中一些最受欢迎的选项包括 CodeSandbox、StackBlitz 和 Replit，这些平台提供了用户友好的界面，并允许在没有任何本地设置的情况下创建、共享和协作 React 项目。

要开始使用在线代码编辑器，甚至不需要账户。只需在浏览器中单击此链接：https://react.new。几秒钟后，将看到 CodeSandbox 已经准备好使用模板项目，并且编辑器的实时预览可以直接在浏览器标签页中获得。如果想要保存更改，则需要创建一个账户。

那么如果用户更喜欢基于浏览器的开发环境，使用在线代码编辑器是学习和尝试 React 的便捷方式。

本节探讨了搭建 React 项目的几种不同方法。无论是选择 Web 打包工具、框架还是在线代码编辑器，每种方法都提供了其独特的优势。读者可以选择自己喜欢的并适合项目需

求的方法。接下来将开始 React 开发之旅。

1.5 本 章 小 结

本章全面介绍了 React，以便读者对它是什么及其必要内容有一个大致的了解，从而为本书的其余部分定下基调。React 是一个用于构建 UI 的库，它的 API 很小。然后介绍了 React 的一些关键概念。我们讨论了 React 的简洁性，因为它没有很多活动部件。

接下来探讨了 React 组件和 JSX 的声明式特性。然后了解到 React 通过编写可以重复渲染的声明式代码来实现高效的性能。

本章还介绍了渲染目标这一概念，以及 React 如何轻松成为各种平台的首选 UI 工具。然后，本章提供了 React 历史的简要概述，并介绍了最新的发展。最后深入介绍了如何搭建一个新的 React 项目并启动学习过程。

目前这些介绍性和概念性的内容已经足够。随着我们继续本书的旅程，我们将重新审视这些概念。第 2 章将讨论 JSX 渲染。