第1章 React 简介

背景介绍

在 Web 应用开发的早期，构建 Web 应用的唯一方案就是向服务器发送请求，然后服务端响应请求并返回一个完整的页面。从开发角度上讲这种方式非常简单，因为开发者无须关心在浏览器端发生了什么。

像 PHP 这类语言更加简化了这种开发方式。使用 PHP 开发功能组件也很容易，这有助于开发者重用代码，掌控应用程序的行为。开发的简单性使得 PHP 成为了一门非常流行的 Web 应用开发语言。

不过，使用这种开发方式很难打造出极佳的用户体验。因为无论每次用户想做点什么，都需要向服务端发送请求并等待服务端的响应，这会导致用户失去在页面上所积累的状态。

为了实现更好的用户体验，人们开始开发类库，使用 JavaScript 在浏览器端渲染应用。这些类库使用的方法也不尽相同，简单的会使用带参数的模板，复杂的就完全掌控整个应用。随着开发者在越来越大的应用中使用这些类库，应用也变得越来越难于把握，因为这些应用是一系列互相作用的事件的结果。与 PHP 那样传统的应用开发方式比起来，这种客户端应用很难做好。

React 发源自 Facebook 的 PHP 框架 XHP 的一个分支。XHP 作为一个 PHP 框架，旨在每次有请求进来时渲染整个页面。React 的产生就是为了把这种重新渲染整个页面的 PHP 式工作流带到客户端应用中来。

React 本质上是一个“状态机”，可以帮助开发者管理复杂的随着时间而变化的状态。它以一个精简的模型实现了这一点。React 只关心两件事：

1. 更新 DOM
2. 响应事件

React 不处理 Ajax、路由和数据存储，也不规定数据组织的方式。它不是一个 Model-View-Controller 框架。如果非要问它是什么，它就是 MVC 里的“V”。React 的精简允许你将它集成到各种各样的系统中。事实上，它已经在数个 MVC 框架中被用来渲染视图了。

在每次状态改变时，使用 JavaScript 重新渲染整个页面会异常慢，这应该归咎于读取和更新 DOM 的性能问题。React 运用一个虚拟的 DOM 实现了一个非常强大的渲染系统，在 React 中对 DOM 只更新不读取。

React 就像高性能的 3D 游戏引擎，以渲染函数为基础。这些函数读入当前的状态，将其转换为目标页面上的一个虚拟表现。只要 React 被告知状态有变化，它就会重新运行这些函数，计算出页面的一个新的虚拟表现，接着自动地把结果转换成必要的 DOM 更新来反映新的表现。

乍一看，这种方式应该比通常的 JavaScript 方案——按需更新每一个元素——要慢，但 React 确实是这么做的：它使用了非常高效的算法，计算出虚拟页面当前版本和新版间的差异，基于这些差异对 DOM 进行必要的最少更新。

React 赢就赢在最小化了重绘，并避免了不必要的 DOM 操作，这两点都是公认的性能瓶颈。

用户界面越复杂，就越容易发生这样的情况——一个用户交互触发一个更新，而这个更新触发另外一个更新，一个接一个。如果没有恰当地把这些更新放到一起的话，性能就会大幅度降低。更糟糕的是，有时候 DOM 元素在达到最终状态前，会被更新好多次。

React 的虚拟表示差异算法，不但能够把这些问题的影响降到最低（通过在单个周期内进行最小的更新），还能简化应用的维护成本。当用户输入或者有其他更新导致状态改变时，我们只要简单地通知 React 状态改变了，它就能自动化地处理剩下的事情。我们无须深入到详细的过程之中。

React 在整个应用中只使用单个事件处理器，并且会把所有的事件委托到这个处理器上。这一点也提升了 React 的性能，因为如果有很多事件处理器也会导致性能问题。

> 我们有一个贯穿全书的示例项目，一个问卷制作工具，你可以在 *https://github.com/backstopmedia/bleeding-edge-sample-app* 阅读全部源码。

本书概览

本书将分四个大块进行讲述，帮助你在开发时充分发挥 React 的优势。

Component 的创建和复合

本书的前 7 章都与 React 组件的创建和复合相关。这些章节将帮助你搞清楚如何使用 React。

第 1 章　React 简介

React 介绍，包括背景介绍及全书概览。

第 2 章　JSX

JSX（JavaScript XML）提供了一种在 JavaScript 中编写声明式的 XML 的方法。这一章将学习如何在 React 中使用 JSX，学习如何构建简单的 React Component。虽然对 React 来说 JSX 不是必需的，但因为这是一种推荐的用法，因此本书的大部分例子都会使用 JSX。

第 3 章　组件的生命周期

在渲染过程中，React 会频繁地创建或者销毁组件。React 提供了很多可被注入到组件生命周期中的钩子函数。你需要了解并理解如何管理组件的生命周期，避免在应用中产生内存泄漏。

第 4 章　数据流

在 React 中，数据是如何在组件树中从上向下传递的？哪些数据可以修改？搞清楚这些问题是非常重要的。React 的 `props` 和 `state` 有明确的区别。这一章将学习 *props* 和 *state* 是什么，以及怎样在应用中正确地使用它们。

第 5 章　事件处理

React 的事件处理采用声明的方式。对于交互式的界面，事件处理是非常重要的一部分，也是必须学习并掌握的。还好 React 提供的事件处理方案非常简单。

第 6 章　组件的复合

React 鼓励创建小巧且有明确功能的组件来处理特定需求，再在应用中创建复合层来组合使用这些组件。这一章将学习如何在其他组件中使用已有的组件。

第 7 章　mixin

mixin 是 React 提供的另外一种在多个 React 组件中共享功能的方式。mixin 还是另外一种将组件拆为更小、更易维护的部分的方式。

进阶

一旦掌握了 React 的基础，就可以继续学习一些高级的主题。接下来的 6 章有助于进一步打磨 React 技巧，搞清楚如何创建优秀的 React 组件。

第 8 章 DOM 操作

尽管 React 提供了基于虚拟 DOM 的各种功能，有时候你还是需要访问应用程序中原生的 DOM 节点。这样就可以利用现有的一些 JavaScript 类库，或者可以更加自由地控制你的组件。本章将告诉你在 React 组件生命周期的哪些节点上可以安全地访问 DOM，在什么时候应该释放对 DOM 的控制，避免内存泄漏。

第 9 章 表单

接受用户输入的最佳方式之一就是使用 HTML 表单。但有一个问题，HTML 表单是有状态的。React 提供了一种方案，可以把大部分状态从表单移入到 React 组件中。这为我们提供了对表单元素的不可思议的控制力。

第 10 章 动画

作为 Web 开发者，我们手里已经有了一个声明式且性能强劲的动画工具：CSS。React 鼓励使用 CSS 实现动画。本章介绍在 React 中如何利用 CSS 给组件添加动画。

第 11 章 性能优化

React 的虚拟 DOM 虽然创造性地提升了性能，但是性能还有继续提升的空间。React 提供了这样一种方式，即当你知道组件没有变化时，可以告诉渲染器无须重新渲染你的组件。通过这种方法可以大幅度地提高应用的速度。

第 12 章 服务端渲染

很多应用都要求进行 SEO，恰好 React 可以像 Node.js 那样在非浏览器环境中渲染。服务端渲染还可以提升应用首页的加载速度。编写同时支持服务端和客户端渲染的应用可能有些困难。本章将提供一些同构渲染的策略，指出在做服务端渲染时，你将碰到哪些具有挑战的关注点。

第 13 章 周边类库

Facebook 开源了 React——其内部使用的所有框架的组成部分。一段时间以来，很多其他的类库也被 Facebook 开源出来，它们可以无缝地与 React 一起使用。这一章将会介绍如何把这些类库与 React 组合到一起。

React 工具

React 有很多很棒的开发工具和测试框架。学会使用这些工具有助于你编写出更健壮的程序。这部分将分成工具和测试两章进行介绍。

第 14 章　开发工具

React 应用变大后，不但需要某种方式自动打包代码进行开发，而且调试程序也变得更加困难。在本章中，你将了解有哪些工具可以用来构建和打包 React 应用，学习如何使用 Google Chrome Plugin 来可视化你的 React 组件，简化调试。

第 15 章　测试

随着应用逐渐变大，为了确保不向已有的可用代码中引入新的问题，编写测试是重要的一部分。因为测试鼓励编写模块化的代码，所以有助于写出更好的代码。本章将带着你学习如何全面地测试 React 组件。

React 实践

最后两章介绍使用 React 时要注意哪些方面，以及其他你可能没有想到的使用场景。

第 16 章　架构模式

React 只提供了“MVC”里面的“V”，但是它非常灵活，可以作为其他框架或者系统的插件使用。本章将带着你学习使用 React 来设计更大规模的应用。

第 17 章　其他使用场景

尽管 React 把注意力放在 Web 上，但是也可以应用在其他支持 JavaScript 的场景下。本章将介绍一些除传统的 Web 之外的使用场景。