

React 全栈

Redux+Flux+webpack+Babel 整合开发



张轩 杨寒星 著

前端撷英馆

React 进阶

Redux+Flux+webpack+Babel 整合开发



张轩 杨寒星 著

电子工业出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书从现代前端开发的标准、趋势和常用工具入手，由此引出了优秀的构建工具 webpack 和 JavaScript 库 React，之后用一系列的实例来阐述两者的特色、概念和基本使用方法。随着应用复杂度的增加，进而介绍了 Flux 和 Redux 两种架构思想，并且使用 Redux 对现有程序进行改造，最后介绍了在开发过程中出现的反模式和性能优化方法。

本书适合有一定前端开发尤其是 JavaScript 基础的读者阅读，如果您还没有接触过前端开发这个领域，请先阅读前端开发的入门书籍。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

React 全栈：Redux+Flux+webpack+Babel 整合开发 / 张轩，杨寒星著. —北京：电子工业出版社，2016.10
（前端撷英馆）

ISBN 978-7-121-29899-8

I. ①R… II. ①张… ②杨… III. ①JAVA 语言—程序设计 IV. ①TP312.8

中国版本图书馆 CIP 数据核字(2016)第 219048 号

策划编辑：张春雨

责任编辑：付 睿

印 刷：北京中新伟业印刷有限公司

装 订：北京中新伟业印刷有限公司

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本：787×980 1/16 印张：14 字数：251 千字

版 次：2016 年 10 月第 1 版

印 次：2016 年 10 月第 1 次印刷

定 价：69.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：(010) 88254888, 88258888。

质量投诉请发邮件至 zlls@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819 faq@phei.com.cn。

前 言

对一个前端工程师来说，这是最坏的时代，也是最好的时代。

在这样的领域里，每一年都不会风平浪静。如果说 2014 年是属于 MVVM，属于 Angular 的，那么 2015 年称为 React 元年并不为过。开发团队的不断完善以及 React 社区井喷式的发展让这个诞生于 2013 年的框架及其生态趋于成熟（就在不久前，React 官方宣布将在版本号 0.14.7 后直接使用版本号 15.0.0），大量团队在生产环境中的实践经验也让引入 React 不再是一件需要瞻前顾后反复调研的事情，如果 React 适合你，那么现在就可以放心地使用了。

可是对于很多还没有深入实践过 React 开发的工程师来说，React 到底做了什么？React 适合什么样的场景？又应该如何投入使用？在具体业务逻辑的实现上，怎样才是最佳的实践？这些都是需要去了解与思考的问题。

本书将从一个传统前端工程师的角度出发，介绍 React 产生的背景及其架构应用，并结合一些由浅入深的例子帮助读者掌握基于 React 的 Web 前端开发方法。

——杨寒星

前端开发是一个充满变化的领域，它的发展速度快得惊人。各种各样的新技术、新标准层出不穷，GitHub 上最火的语言是 JavaScript，最大的包管理器是 npm。新的流行框架日新月异，几年前的那些先驱者还是工程师口中津津乐道的宠儿，比如 YUI、Mootools、jQuery 等，今天已经不再那么流行，曾经名噪一时的 Backbone 框架，现

在也渐渐褪去热度，继往开来的 Angular、Vue.js、Ember 等 MVVM 框架竞相登场，再加上当红的新宠 **React.js** 大行其道，让好多工程师仿佛迷失在了大潮中。

前端开发是一个新兴的行业。几年前，被称作重构工程师的我们还都在对着 Photoshop 切图，把一些 jQuery 插件复制来复制去，完成一些炫酷的幻灯图特效，不断地处理着很多 IE 浏览器的怪异 Bug。这些功力其实到现在还能满足大部分的 Web 开发，完成大部分的项目。我们不妨把它称为“古典时代”，它影响深远，但是最终会慢慢远去。

在当前这个潮流下，很多工程师会抛出这样的言论：

学习一些新的工具、框架有什么用？业界发展得这么快，等我学会了这些，它也许已经“寿终正寝”了。天天跟风一样地追求各种框架，学会了也是迷茫，这些框架没有用武之地。旁门左道，天天布道没有用的东西，伪前端。

随着技术的进化、移动应用的飞速发展，一个前端工程师的职责不像原来那样只要把图转换成网页那么简单。如今产生了各种类型的新名词——Hybird 应用、全端工程师、SPA 等，各有其特定的应用场景。任何框架的发明和创造都有它们一定的历史原因，只有解决了需求的痛点，才能让工程师更快地解决难题。在我们学习的过程中，可以发现它背后的思想和解决方案，进而更好地充实自己。做技术的人最重要的就是保持开放的态度，有一颗好奇心，持续不断地学习。

在前端开发中占最重要部分的 JavaScript，也随着这些框架在慢慢进化着，原来令人不断诟病的缺点正在被标准制定者慢慢修补，新的特性不断浮出水面。前端工程师正处在发展最迅速的时代，这应该是一个让人兴奋的时代，犹如工业革命一样，每个工程师都见证着一场伟大的前端革命。

本书不仅讲述了怎样使用 React 和 webpack 开发一些应用，而且希望通过一系列的介绍让每个工程师都能站在前端技术的潮头，拥抱变化，学习新的标准和技术，成为新技术的弄潮儿。

——张轩

本书面向的读者

本书适合有一定前端开发经验尤其是有 JavaScript 基础的读者，如果你还没有接触过前端开发这个领域，请先阅读前端开发的入门书籍。

本书的代码示例

你可以在这里下载本书的代码示例：<https://github.com/vikingmute/webpack-react-codes>。

本书的代码执行环境

本书中默认的开发环境是 Node.js 5.0.0，书中介绍到的几个库的版本分别为 React@15.0.1、webpack@1.12.14 及 Redux@3.2.1，其他如未特别说明的则为最新版本。

目 录

第 1 章 现代前端开发	1
1.1 ES6——新一代的 JavaScript 标准	1
1.1.1 语言特性	2
1.1.2 使用 Babel	10
1.1.3 小结	13
1.2 前端组件化方案	13
1.2.1 JavaScript 模块化方案	14
1.2.2 前端的模块化和组件化	16
1.2.3 小结	18
1.3 辅助工具	19
1.3.1 包管理器 (Package Manager)	19
1.3.2 任务流工具 (Task Runner)	23
1.3.3 模块打包工具 (Bundler)	26
第 2 章 webpack	28
2.1 webpack 的特点与优势	28
2.1.1 webpack 与 RequireJS、browserify	29
2.1.2 模块规范	30

2.1.3	非 javascript 模块支持	31
2.1.4	构建产物	32
2.1.5	使用	33
2.1.6	webpack 的特色	35
2.1.7	小结	38
2.2	基于 webpack 进行开发	38
2.2.1	安装	38
2.2.2	Hello world	39
2.2.3	使用 loader	43
2.2.4	配置文件	46
2.2.5	使用 plugin	48
2.2.6	实时构建	50
第 3 章	初识 React	52
3.1	使用 React 与传统前端开发的比较	54
3.1.1	传统做法	54
3.1.2	全量更新	56
3.1.3	使用 React	57
3.1.4	小结	59
3.2	JSX	59
3.2.1	来历	59
3.2.2	语法	60
3.2.3	编译 JSX	63
3.2.4	小结	64
3.3	React+webpack 开发环境	64
3.3.1	安装配置 Babel	64
3.3.2	安装配置 ESLint	65
3.3.3	配置 webpack	66

3.3.4	添加测试页面	68
3.3.5	添加组件热加载 (HMR) 功能	70
3.3.6	小结	71
3.4	组件	72
3.4.1	props 属性	73
3.4.2	state 状态	76
3.4.3	组件生命周期	78
3.4.4	组合组件	80
3.4.5	无状态函数式组件	82
3.4.6	state 设计原则	82
3.4.7	DOM 操作	83
3.5	Virtual DOM	85
3.5.1	DOM	85
3.5.2	虚拟元素	86
3.5.3	比较差异	88
第 4 章	实践 React	91
4.1	开发项目	91
4.1.1	将原型图分割成不同组件	92
4.1.2	创造每个静态组件	93
4.1.3	组合静态组件	96
4.1.4	添加 state 的结构	99
4.1.5	组件交互设计	100
4.1.6	组合成为最终版本	102
4.1.7	小结	105
4.2	测试	106
4.2.1	通用测试工具简介	106
4.2.2	React 测试工具及方法	108

4.2.3	配置测试环境	109
4.2.4	Shallow Render	110
4.2.5	DOM Rendering	114
4.2.6	小结	116
第 5 章	Flux 架构及其实现	117
5.1	Flux	117
5.1.1	单向数据流	118
5.1.2	项目结构	119
5.1.3	Dispatcher 和 action	119
5.1.4	store 和 Dispatcher	122
5.1.5	store 和 view	124
5.1.6	Flux 的优缺点	126
5.1.7	Flux 的实现	126
5.2	Redux	126
5.2.1	动机	127
5.2.2	三大定律	127
5.2.3	组成	129
5.2.4	数据流	136
5.2.5	使用 middleware	137
第 6 章	使用 Redux	142
6.1	在 React 项目中使用 Redux	142
6.1.1	如何在 React 项目中使用 Redux	142
6.1.2	react-redux	147
6.1.3	组件组织	152
6.1.4	开发工具	155

6.2	使用 Redux 重构 Deskmark	157
6.2.1	概要	157
6.2.2	创建与触发 action.....	158
6.2.3	使用 middleware	159
6.2.4	实现 reducer.....	163
6.2.5	创建与连接 store	165
第 7 章	React+Redux 进阶	168
7.1	常见误解	168
7.1.1	React 的角色.....	169
7.1.2	JSX 的角色	169
7.1.3	React 的性能.....	170
7.1.4	“短路”式性能优化.....	171
7.1.5	无状态函数式组件的性能	172
7.2	反模式	173
7.2.1	基于 props 得到初始 state.....	173
7.2.2	使用 refs 获取子组件.....	176
7.2.3	冗余事实	178
7.2.4	组件的隐式数据源.....	180
7.2.5	不被预期的副作用.....	182
7.3	性能优化	183
7.3.1	优化原则	183
7.3.2	性能分析	184
7.3.3	生产环境版本	187
7.3.4	避免不必要的 render	188
7.3.5	合理拆分组件	199
7.3.6	合理使用组件内部 state.....	200
7.3.7	小结.....	203

7.4 社区产物	203
7.4.1 Flux 及其实现.....	203
7.4.2 Flux Standard Action	204
7.4.3 Ducks	206
7.4.4 GraphQL/Relay 与 Falcor	207
7.4.5 副作用的处理	209

第 1 章 现代前端开发

前端开发现在在经历急速发展的阶段。随着应用场景越来越广，需求越来越复杂，社区和官方也在不断地将其规范化和工程化。在本章中，会给读者介绍目前前端开发的发展现状和一些优秀的工具。如果你还在使用比较古老的开发方式，那么不妨跟随步伐，了解一下现阶段的发展。

本章会从 3 个方面介绍现代的前端开发技术，作为了解 React 和 webpack 的背景知识。

- ES6——新一代的 JavaScript 语言标准。
- Component 组件和模块的发展历程。
- 前端开发的常用工具：
 - 包管理器（Package Manager），用来下载和管理前端代码库。
 - 任务流工具（Task Runner），用来执行一系列开发中的任务。
 - 模块打包工具（Bundler），用来转换和合并前端代码的模式。

1.1 ES6——新一代的 JavaScript 标准

JavaScript 这门脚本语言一直被人诟病（所以薄薄的一本《JavaScript 语言精粹》让很多读者奉为圣经），再加上浏览器兼容性的问题，令很多前端开发者感到特别苦

恼。如今前端开发发展又如此迅速，这促使了 **ECMA** 组委会在修订 JavaScript 语言新版本时，不仅在质量上不断加以完善，同时加快了更新的速度。

ES6（或者被称为 ES2015）被称为 JavaScript 历史上最重大的一次变革，该标准最终敲定于 2015 年 6 月，提供了非常多语言级别新的特性。这是一个可以载入前端发展史册的重大事件。本书全面使用了 ES6 标准，在这里会简单描述一些在本书中使用的 ES6 的特性，给读者一个关于最新标准的直观概念。如果想了解更多变化和特性，建议阅读阮一峰老师编写的《ECMAScript 6 入门》。

1.1.1 语言特性

1. const、let 关键字

众所周知，在 JavaScript 中，变量默认是全局性的，只存在函数级作用域，声明函数曾经是创造作用域的唯一方法。这点和其他编程语言存在差异，其他语言大多数都存在块级作用域。所以在 ES6 中，新提出的 **let** 关键字使这个缺陷得到了修复。

```
if (true) {
  let a = 'name';
}
console.log(name);
// ReferenceError: a is not defined
```

同时还引入的概念是 **const**，用来定义一个常量，一旦定义以后不可以修改，不过如果是引用类型的，那么可以改变它的属性。

```
const MYNAME = 'viking';
MYNAME = 'kitty';
// "CONSTANT" is read-only
const MYNAME = {foo: 'viking'};
MYNAME.foo = 'kitty';
//可以正常运行
```

2. 函数

- 箭头函数

箭头函数是一种更简单的函数声明方式，可以把它看作是一种语法糖，箭头函

数永远是匿名的。

```
let add = (a, b) => {return a + b;}
//当后面是表达式(expression)的时候, 还可以简写成
let add = (a, b) => a + b;
//等同于
let add = function(a, b) {
  return a + b;
}
//在回调函数中应用
let numbers = [1, 2, 3];
let doubleNumbers = numbers.map((number) => number * 2);
console.log(doubleNumbers);
//[2, 4, 6] 看起来很简便吧
```

- **this** 在箭头函数中的使用

在工作中经常会遇到这样的问题, 就是 **this** 在一个对象方法中嵌套函数。

```
let age = 2;
let kitty = {
  age: 1,
  grow: function() {
    setTimeout(function() {
      console.log(++this.age);
    }, 100);
  }
};

kitty.grow();
// 3
```

在对象方法的嵌套函数中, **this** 会指向 **global** 对象, 这被看作是 JavaScript 在设计上的一个重大缺陷, 一般都会采用一些 **hack** 来解决它, 如下。

```
let kitty = {
  age: 1,
  grow: function() {
    const self = this;
    setTimeout(function() {
      console.log(++self.age);
    }, 100);
  }
}
```

```
}  
//或者  
let kitty = {  
  age: 1,  
  grow: function() {  
    setTimeout(function() {  
      console.log(this.age);  
    }).bind(this, 100);  
  }  
}
```

现在有了箭头函数，可以很轻松地解决这个问题。

```
let kitty = {  
  age: 1,  
  grow: function() {  
    setTimeout(() => {  
      console.log(this.age);  
    }, 100);  
  }  
}
```

- 函数默认参数

ES6 没有出现之前，面对默认参数都会让人感到很痛苦，不得不采用各种 hack，比如说：`values = values || []`。现在一切都变得轻松很多。

```
function desc(name = 'Peter', age = 5) {  
  return name + ' is ' + age + ' years old';  
}  
desc();  
//Peter is 5 years old
```

- Rest 参数

当一个函数的最后一个参数有“...”这样的前缀，它就会变成一个参数的数组。

```
function test(...args) {  
  console.log(args);  
}  
test(1, 2, 3);  
// [1, 2, 3]  
function test2(name, ...args) {  
  console.log(args);  
}
```



```

}
test2('Peter', 2, 3);
//[2, 3]

```

它和 `arguments` 有如下区别：① Rest 参数只是没有指定变量名称的参数数组，而 `arguments` 是所有参数的集合；② `arguments` 对象不是一个真正的数组，而 Rest 参数是一个真正的数组，可以使用各种方法，比如 `sort`、`map` 等。有了这两个理由，是时候告别 `arguments`，拥抱可爱的 Rest 参数了。

3. 展开操作符

刚才在函数中讲到了使用“...”操作符来实现函数参数的数组，其实这个操作符的魔力不仅仅如此。它被称为展开操作符，允许一个表达式在某处展开，在存在多个参数（用于函数调用）、多个元素（用于数组字面量）或者多个变量（用于解构赋值）的地方就会出现这种情况。

- 用于函数调用

如果在之前的 JavaScript 中，想让函数把一个数组依次作为参数进行调用，一般会如下这样做。

```

function test(x, y, z) { };
var args = [1, 2, 3];
test.apply(null, args);

```

有了 ES6 的展开运算符，可以简化这个过程。

```

function test(x, y, z) { };
let args = [0, 1, 2];
test(...args);

```

- 用于数组字面量

在之前的版本中，如果想创建含有某些元素的新数组，常常会用到 `splice`、`concat`、`push` 等方法，如下。

```

var arr1 = [1, 2, 3];
var arr2 = [4, 5, 6];
var arr3 = arr1.concat(arr2);
console.log(arr3);

```