React.js 实战

Web 前端技术丛书

快速学会React，全面掌握Web前端开发技术

- 针对每个知识点使用示例进行说明，更加注重实践
- 简易笔记本、购物车2个项目案例，快速提升实战能力
- 适合所有想全面掌握React前端开发技术的人员阅读

赵荣娇　刘江虹　著

示例源代码

清華大學出版社

React.js 实战

Web 前端技术丛书

赵荣娇　刘江虹　著

清華大學出版社
北京

内 容 简 介

本书旨在帮读者从零开始学习 React 基础知识，采用“语法”+“示例”的方式，以便于初学者学习和练习，是目前市场上少有的 React 入门图书。

本书共 14 章，分为 3 篇，涵盖的主要内容有：React 的前世今生、使用 React 所需的预备知识（包括 npm、webpack、ES6）、React 开发环境搭建、React 组件、React 事件系统、React 原理、数据管理、React 架构、React 服务端渲染、React 测试、React 性能优化、React+webpack+ES6 项目实战（笔记本+购物车）等。

本书内容丰富、实例典型、实用性强，适合有一定的 HTML、CSS、JavaScript 基础、希望全面学习 React 开发的前端开发人员阅读，也适合希望提高项目开发水平的人员阅读。

图书在版编目（CIP）数据

React.js 实战 / 赵荣娇，刘江虹著.—北京：清华大学出版社，2019
（Web 前端技术丛书）
ISBN 978-7-302-52873-9

Ⅰ. ①R… Ⅱ. ①赵… ②刘… 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 083029 号

责任编辑：夏毓彦
封面设计：王　翔
责任校对：闫秀华
责任印制：李红英

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：三河市龙大印装有限公司
经　　销：全国新华书店
开　　本：190mm×260mm　　**印　　张**：17.25　　**字　　数**：442 千字
版　　次：2019 年 6 月第 1 版　　**印　　次**：2019 年 6 月第 1 次印刷
定　　价：59.00 元

产品编号：082471-01

前　言

随着互联网技术的发展，前端技术的发展也进入一个新的阶段。早期的网页开发是由后端主导的，前端的操作局限于 DOM 区域。随着基础设置的不断完善以及代码封装层级的不断提高，使得前端能够完成的事越来越多，前端所需解决的业务场景也越来越复杂。

近几年，前端已经发展到跨端、跨界面的革新阶段，目前主流以基于 MVVM、Virtual DOM、前后端同构技术进行开发的项目居多，实现的方向也多种多样。React 就是在此基础上发展起来的框架，独特的设计思想所带来的革命性创新让其成为前端新技术的代表。

目前市场上关于 React 开发及实践的图书不少，但真正从零基础搭建开始，通过语法和小示例指导读者提高开发水平的图书却很少。本书便是以实战为主旨，通过 React 开发中所需要涉及的基础知识和两个完整的项目案例，让读者全面、深入、透彻地理解 React 开发的技术栈的整合使用。

本书的技术点

本书涵盖 npm、Node.js、webpack、ES6、React、JSX、Redux、Jest、Enzyme、Hooks、ESLint、Chrome 插件、JavaScript、CSS、ImmutableJS、Perf 等热门技术及整个技术站框架的整合使用。

本书最后使用 React+webpack+ES6 组合形式，开发了笔记本和购物车两个完整项目。读者将案例稍加修改，便可用于实际项目开发实践中。

本书的内容

本书共有 14 章，由浅到深地介绍 React 技术栈中的主要技术，主要内容分为基础篇、进阶篇和实战篇，每一篇内容又分成若干章节来介绍。

第 1 篇　基础篇（第 1～3 章）

介绍 React 的前世今生，以及 React 开发中涉及的基本概念，包括 React 的开发环境和开发工具、React 的基本用法。每个知识点都有配套的源代码示例。

第 2 篇　进阶篇（第 4～12 章）

深入介绍 React 的几个重要概念，包括 React 组件、React 事件系统、React 原理、数据管理、React 架构、React 服务端渲染等。每章都配有大量示例代码，保证读者学以致用。

第 3 篇　实战篇（第 13～14 章）

本篇通过笔记本和购物车两个项目整合使用 React 技术栈，包括 React Router、Redux、SSR，每一个技术都配有详细的项目实战演示。

关于封面照片

封面照片由蜂鸟网的摄影家 ptwkzj 先生友情提供，在此表示衷心感谢。

读者对象

- 有一定的 HTML、CSS、JavaScript 基础的网页开发人员；
- 希望全面学习 React 开发的前端开发人员；
- 希望提高项目开发水平的人员；
- 前端开发培训机构的学员；
- 软件开发项目经理；
- 需要一本案头必备查询手册的人员。

本书作者

本书第 1~6 章由刘江虹完成，第 7~14 章由赵荣娇完成。

著　者

2019 年 5 月

目 录

第 1 章

◂React的前世今生▸

时间追溯到 20 世纪 90 年代，网景浏览器的诞生给互联网世界填写了浓墨的一笔，用户可以在浏览器上查阅信息、分享信息，以及和浏览器进行交互等，当时网景浏览器在市场上的份额是占据第一的。后来微软为了瓜分市场，推出了 IE 浏览器，凭借 Windows 系统的用户量把网景打败。就在微软觉得稳坐天下的时候，Firefox 以及 Chrome 等优秀浏览器的出现，打破了 IE 的市场垄断。可见产品是需要不断创新、不断成长的，故步自封终究会被超越。

JavaScript 的运行环境就是浏览器，其实前端框架也一直处在一个纷争的世界。目前市面上各种 xxx.js 的出现，包括 Ember.js、Angular.js、Backbone.js、Knockout.js、React.js、Vue.js 等，也是纷纷扰扰。前端开发在目前这个互联网环境下还是一个大的趋势，以 JavaScript 为基础语言的前端框架层出不穷，前端技术的更新也非常之快。当今前端框架格局，就上述提到的框架，可以说 Angular、Vue 和 React 三足鼎立。本书主要介绍 React。

1.1 刀耕火种的年代

所谓刀耕火种，就是在 Web 开发早期，技术和工具还不成熟，Web 开发功能和用户体验非常有限，Web 开发人员使用古老的工具耕耘着 Web 这片广阔的天地。

追溯到 Web1.0 时代，当时的 Web 整体架构非常简单，页面基本由 Server 端生成，然后返回给浏览器，页面的呈现也特别粗糙。最初 Web 开发是不分前后端的，所有的逻辑都是在服务器端生成。如果业务逻辑比较简单，那这种方式对于开发人员来说是可以接受的。如果业务逻辑非常复杂，这样会出现很多问题，最大的一个缺点就是关系如果变得复杂，就会导致 Server 非常臃肿，条理不清晰。

直到 MVC 时代的到来，以后端作为出发点，开始细化模块功能，这个时代催生了一些经典的 MVC 框架，比如 Structs、Spring 等。M（Model）层负责数据处理，V（View）层负责界面呈现，C（Controller）层负责处理用户交互功能。这种模式的优点在于开发人员可以各司其职，互不干涉，分工明确。当然也有缺点，View 层和 Controller 层黏度很高，会增加 Web 开发的复杂度。

1.2 Web 应用的出现

有了 Web2.0 这个概念后，互联网开始进入一个新时代。以前用户在浏览器上只能接收文字、图片这样的资讯，处于一个被动接收的角色。在进入 Web2.0 时代后，Web 和用户的交互性加强，类似于桌面程序的 Web 应用大量涌现，这个时期 ajax 的诞生拉开了 SPA（Single Page Application，单页面应用）序幕。

另外，多媒体的诞生也催生了 Web2.0 时代的发展，像音频、视频、Flash 的出现，可以让网页变得更加绚丽多彩，给用户在视觉和听觉上都带来了不一样的体验。可以说 Web2.0 时代的到来给 Web 前端这一块带来了空前的繁荣。

1.3 React 的诞生

React 出生在 Facebook。当初 Facebook 要搭建一个 Instagram 网站，在选择框架时，对市场上的所有 JavaScript 的 MVC 框架都不太满意，于是 Facebook 自己搞了一套，就是 React。后来发现 React 框架非常好用，便在 2013 年 5 月开源了。

React 的出现给 Web 前端开发人员带来了福音，其新颖的创新思路，加上极佳的性能，广受前端开发人员的欢迎。下一章将从零开始讲解 React 的环境搭建、使用方法以及如何设计的相关知识。

在学习 React 之前，读者应该对以下知识进行了解：

- Node.js：基于 Chrome V8 引擎的 JavaScript 运行环境，使用了一个事件驱动、非阻塞 I/O 的模式，使其轻量又高效。
- npm：node 的一个包管理工具，主要功能是对 node 包的安装、卸载、更新、查看等。
- webpack：前端资源加载/打包工具，可以依据模块的依赖关系进行静态分析，按照一定规则将这些模块生成静态资源。
- ES6：也称为 ES2015，是在 2015 年 6 月发布的 JavaScript 语言的下一代标准，旨在编写大型应用程序，成为一种企业级开发语言。

如果读者对 ES5 比较熟悉，可以使用 ES5 进行 React 开发，但是 React 推荐使用 ES6，这在以后是一个趋势，目前 Facebook 官方推荐的标准是 ES6。

1.4 npm

在使用任何一个框架之前，必然要经历的一个环节是环境搭建，而 npm 是配置 React 环境的必要工具，其在下载各种依赖包时起着重要作用。本节主要为读者解析 npm 是怎样的一个工具。

1.4.1 什么是 npm

简单来说 npm（Node Package Manager）是包含在 Node.js 里面的一个包管理工具，如果读者之前使用过 Node.js，那对 npm 应该不会陌生，因为 npm 会随着 Node.js 一起安装。npm 是世界上最大的软件注册表，其为开发者连接到了一个广阔的 JavaScript 世界。据官方数据统计，npm 大约每周有 30 亿的惊人下载量，其中包含大约 60 万个 package（代码模块）。

npm 为开发者提供了一个代码模块共享的大平台，开发者既可以从 npm 服务器上下载其他开发人员共享的 package，也可以上传自己的 package 供其他开发者使用。Node.js 和 npm 的环境搭建将在 2.2 节中详细讲述。

1.4.2 理解 npm scripts

npm scripts 指的是 npm 脚本，其主要用途是执行配置文件（package.json）中“scripts”属性对应的脚本语句。在理解 npm scripts 之前，这里先介绍一下 package.json 文件。

在搭建一个前端项目时，一般在项目的根目录下要生成一个 package.json 文件，该文件用来定义项目信息、配置包依赖关系。package.json 文件可以自己手动创建，也可以用如下命令创建：

```
$ npm init
```

这里列举一个简单的 package.json 文件，如下所示：

```
{
  "name": "ReactDemo",
  "version": "0.1"
}
```

上述 package.json 文件只定义了项目名称和项目版本号。一般情况下，在实际开发中，package.json 文件是非常丰富的，接下来列举一个实际开发中比较全面的 package.json 文件，如下所示：

```
{
  "name": "demo01",
  "version": "0.1.0",
```

```
  "private": true,
  "dependencies": {
    "React": "^16.2.0",
    "React-dom": "^16.2.0",
    "React-scripts": "1.1.1"
  },
  "scripts": {
    "start": "React-scripts start",
    "build": "React-scripts build",
    "test": "React-scripts test --env=jsdom",
    "eject": "React-scripts eject"
  },
  "devDependencies": {
    "prop-types": "^15.6.1"
  }
}
```

上述 package.json 文件内容多了几个字段，private、dependencies、devDependencies 和 scripts。private 指包的私有属性，如果设置为 true，则 npm 会拒绝发布，主要是为了避免私有 repositories 不小心被发布出去。dependencies、devDependencies 两个字段在 1.4.3 小节介绍，这里主要介绍 scripts。

scripts 里面放的是 npm 要执行的命令，格式是 key-value 形式，为了简化操作，具体命令为 value，自定义的简化命令为 key，当 npm 运行 key 命令时，等同于执行后面的 value 命令。例如，执行 npm run start 命令，相当于执行了 React-scripts start。

简而言之，package.json 配置文件中的脚本，就叫作 npm scripts。其实 scripts 里面的命令可以是任何的 shell 命令，执行 npm run 的时候，会自动构建一个 shell，脚本都是在 shell 中执行，所有 package.json 中的脚本可以是任何可以在 shell 中有效执行的命令。

也许有读者会有疑问，为什么 scripts 命令可以直接使用。这里解释一下，npm run 在新建一个 shell 的时候，会将当前目录的 node_modules/.bin 目录配置到 path 环境变量中，如果以前用过 Java，应该了解在配置 jdk 时需要将 jdk 目录配置到 path 环境变量中才可以全局使用。这里 npm 是自动将 node_modules/.bin 配置到了环境变量中。其实上面的 scripts 字段可以改写为下面的样子：

```
"scripts": {
    "start": "./node_modules/.bin/React-scripts start",
    "build": "./node_modules/.bin/React-scripts build",
    "test": "./node_modules/.bin/React-scripts test --env=jsdom",
    "eject": "./node_modules/.bin/React-scripts eject"
  },
```

1.4.3　dependencies 和 devDependencies

在 1.4.2 小节中介绍 package.json 文件的内容时提到了 dependencies 和 devDependencies 两个字段：

```
"dependencies": {
    "React": "^16.2.0",
    "React-dom": "^16.2.0",
    "React-scripts": "1.1.1"
  },
//…
"devDependencies": {
    "prop-types": "^15.6.1"
  }
```

dependencies 和 devDependencies 两个配置字段都是用来安装依赖包的，区别在于前者安装项目运行所依赖的模块，后者安装项目开发所依赖的模块。

在 npm 安装模块的时候，会有两个命令：

```
$ npm install <package-name> --save
```

和

```
$ npm install <package-name> --save-dev
```

第一个命令是用来对应 dependencies 字段的，第二个命令是对应 devDependencies 字段的。上述示例中有些模块的版本号之前有个插入号“^”，比如"React": "^16.2.0"，表示安装 React 的 16.x.x 的最新版本（不低于 16.2.0），但是不安装 17.x.x，也就是说安装时不改变大版本号。如果版本号前面没有任何标识，比如"React-scripts": "1.1.1"，表示只安装 React-scripts 的 1.1.1 版本。

有时版本号前面会有波浪“~”，例如“~2.2.3”，表示安装 2.2.x 的最新版本号（不低于 2.2.3），但是不安装 2.3.x，也就是说安装时不改变大版本号和次版本号。另外，需要注意的是，如果大版本号为 0，“~”和“^”的表示作用是一样的。

1.5　webpack

在没有出现模块管理器之前的前端开发，如果要引用依赖资源，通常的做法是将依赖文件引用到.html 文件中。比如，要引用 js 文件，在.html 文件中用<script>标签引用；引用.css 文件，在.html 文件中用<link>标签引用。这样做的弊端是，如果引用的资源文件太多，请求太多，会拖慢网页加载速度，影响用户体验，另外也会使网页体积臃肿、不便维护。随着模块管理器

的出现，上述问题得到解决。目前市面上流行的包管理器有很多，比如 Bower、Browserify、webpack 等，本书主要讲解 webpack 这个前端的包管理器，其他工具的用法和 webpack 大同小异，读者自行网上学习即可。

之前引用的.js 文件或.css 文件即可理解为模块。

webpack 主要有 4 个核心概念：

- 入口（entry）：webpack 所有依赖关系图的起点。
- 出口（output）：指定 webpack 打包应用程序的地方。
- 加载器（loader）：指定加载的需要处理的各类文件。
- 插件（plugins）：定义项目要用到的插件。

1.5.1 为什么需要 webpack

移动互联网时代的网站正在慢慢演化为 Web APP，这种局势愈演愈烈，读者应该能感受到 Web 前端发展之迅速，浏览器在不断强大，JavaScript 的新标准 ES6 也在 2015 年制定，各种流行的 JS 框架问世，发展着实快。另外，现在的 Web 前端更倾向单页面应用（SPA），要求的页面刷新越来越少，这样庞大的代码量如果管理不好就会导致很多的后续问题，比如各个模块耦合度变高、很难维护等。这样就催生了模块管理器。

webpack 是前端的一个模块管理工具，其可依据各个模块之间的依赖关系进行静态分析，然后将这些模块按照相应规则生产静态资源供项目调用。可以通过图 1-1 来理解 webpack 是做什么的。

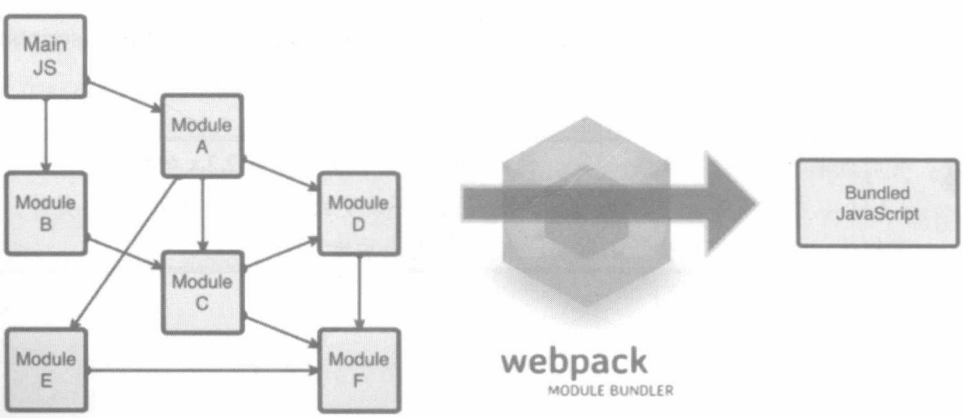

图 1-1　webpack 示意图

可以看出，webpack 可以将具有各种依赖关系的静态模块转化成一个独立的静态模块，这样做的好处是大大减少了请求次数，提高了网页的性能，用户体验也随之提高。

webpack 的另一个作用是可以把目前一些浏览器解释不了的语言进行编译，转换成浏览器可以识别的内容。React 的所有代码示例都以 ES6 标准讲解，ES6 的有些语法目前在一些主流

的浏览器上还不支持，需要对 webpack 进行一些配置后，React 才可正常运行。

1.5.2　webpack 入口和出口

webpack 的四大要素即 entry、output、loader 和 plugins，在讲解这几个要素之前，先了解怎么安装 webpack。

webpack 的安装需要 npm 来完成，有两种安装方式，命令如下：

```
$ npm install -g webpack              //全局安装
```

或者

```
$ npm install --save-dev webpack          //安装到项目目录中
```

安装好之后，就可以使用 webpack 的命令了。比如现在有一个 main.js 文件，要将其打包生成一个 bundle.js 文件，就可以用下面的命令：

```
$ webpack main.js bundle.js
```

一般在实际的项目开发中，要把这些命令写到一个名为 webpack.config.js 的文件中。

webpack 需要处理具有依赖的各个模块，这些模块会构成一个关系图。webpack 的入口就是这张关系图的起点，指的就是入口文件。webpack 的出口指的是需要把这张关系图导出到哪个文件中，即导出文件。这里以一个具体的 webpack.config.js 文件讲解，配置如下：

```
module.exports = {
  entry: './main.js',
  output: {
    filename: 'bundle.js'
  }
};
```

上述 webpack.config.js 文件只配置了项目的 entry 和 output。在该项目的关系图中，main.js 是起点，main.js 可能和别的模块存在依赖关系，但是开发者不需要关心这些，寻找依赖、解决依赖是 webpack 的工作。

entry 字段指定入口文件，也可以理解为 APP 启动时运行的第一个文件。其语法如下：

```
entry: string | Arrary<string>
```

entry 字段可以为一个字符串，也可以是一个字符串数组，所以 entry 可以指定一个入口文件，也可以指定多个入口文件。

output 主要是告诉 webpack 把整理后的所有资源都放在哪里，指定输出位置。上述示例中，output 只指定了 filename，其实还可以指定路径 path，如果省略 path 参数，将默认输出到 webpack.config.js 同级目录下。