学校代码 10126 学号 0141122427

分 类 号 密级

本科毕业论文

|  |
| --- |
| 基于React、Redux和Webpack的前端优化方案 |

学院、系 计算机（软件）学院软件工程系

专业名称 软件工程

年 级 2014级

学生姓名 史安琪

指导教师 邢 熠

2018年4月18日

基于React、Redux和Webpack的前端优化方案

摘 要

随着近几年来互联网技术的极速发展，人们对它的依赖也越来越强，和计算机进行交互的场景也越来越多，随之而来的是公众对用户体验的要求越来越高。前端开发技术的发展能够大大简化开发者的开发难度，从而减少开发者的开发成本并且网站的整体质量。本文主要探讨在前端开发领域方面的应用，运用React、Redux和Webpack的深度整合，来优化网页的性能和质量，在更小的打包体积下极大提升网站的响应速度和兼容性。论文主要包括使用React进行组件化开发，使用Redux管理复杂的单页Web应用（SPA）中的换数据和运行状态，并且使用Webpack对项目进行打包优化。之后根据以上理论构建出一个SPA云音乐客户端作为原型来研究在具体项目中通过React、Redux和Webpack来优化网站的整体性能。最终经过打包后的程序可以减少50%以上的大小，网络请求的数量可以从几十个下降到几个，整个的网站的响应速度可以有3-4秒的提升。

关键词：React, Redux, Webpack, 前端开发，网站优化

Front-end optimization solution based on React, Redux and Webpack

Author: SHI Archie

Tutor: XING Yi

ABSTRACT

With the rapid development of Internet technology in recent years, people’s reliance on it has become stronger and more and more interactions with computers have taken place. As a result, the public’s requirements for user experience have become higher and higher. The development of front-end development technology can greatly simplify the development of developers, thereby reducing the developer's development costs and the overall quality of the site. This article mainly discusses the application in the front-end development field, and uses the deep integration of React, Redux, and Webpack to optimize the performance and quality of the webpage, greatly improving the responsiveness and compatibility of the web site under a smaller package size. The thesis mainly includes component development using React, using Redux to manage the changing data and running status in a complex single-page Web application (SPA), and using Webpack to package and optimize the project. Based on the above theory, a SPA cloud music client was built to study the overall performance of the website through React, Redux, and Webpack in specific projects. After the final packaged program can reduce the size of more than 50%, the number of network requests can be reduced from a few dozen to several, the entire site's response speed can be improved 3-4 seconds.

Key words: React, Redux, Webpack, Front End Development, Website Optimization

目 录

[1.绪论 1](#_Toc513199658)

[1.1选题背景及研究意义 1](#_Toc513199659)

[1.2论文主要内容 2](#_Toc513199660)

[2.相关技术分析 3](#_Toc513199661)

[2.1 React 3](#_Toc513199662)

[2.2 Redux 4](#_Toc513199663)

[2.3 Webpack 5](#_Toc513199664)

[3.原型设计及实现 7](#_Toc513199665)

[3.1总体架构设计 7](#_Toc513199666)

[3.1.1原型搭建 7](#_Toc513199667)

[3.1.2用babel兼容最新版JavaScript规范 8](#_Toc513199668)

[3.1.3用eslint & stylelint对代码静态检查 8](#_Toc513199669)

[3.2原型的优化方案 9](#_Toc513199670)

[3.2.1 CSS样式优化 9](#_Toc513199671)

[3.2.2图标优化 9](#_Toc513199672)

[3.2.3 Webpack性能优化 12](#_Toc513199673)

[3.3原型自动化测试及持续集成 13](#_Toc513199674)

[3.3.1自动化测试 13](#_Toc513199675)

[3.3.2持续集成 14](#_Toc513199676)

[4.优化效果评估 15](#_Toc513199677)

[4.1 lighthouse测试 15](#_Toc513199678)

[4.2打包大小 15](#_Toc513199679)

[结论 18](#_Toc513199680)

[参考文献 19](#_Toc513199681)

# 绪论

## 1.1选题背景及研究意义

互联网的快速发展使人们对它的依赖也变得越来强，不论是在科研机构，商业领域还是个人生活，我们无时无刻不在直接或者间接的与计算机进行交互。然而随着业务需求逐渐繁杂，用户需要的界面和数据的交互也越来越繁杂。编写传统网页的复杂度越来越大，维护成本也随之成倍的提高，往往一个小小的改动就要花费很长的时间去修改。此外各个企业之间的竞争越来越大，为了提升用户体验，并且应对多变的市场环境，各个公司也在不断探寻更好的前端技术。前端技术的发展程度关系到各行各业共同的利益，也关系到我们每个人的上网体验。所有的这些方面，都促使前端技术必须不断的发展，来适应日益增长的需求。

前端技术的发展，从1991年Tim Berners-Lee公开提及HTML，到1994年Netscape公司推出了Navigator1.0，再到1995年JavaScript横空出世，使得计算也可以在网页上进行。之后在1999年W3C组织了正式发布HTML4.0版本，网页技术变的越来越成熟。与此同时，互联网的规模也在逐渐扩张。

之后是2006年8月，jQuery1.0版本进入大家视野，因为其语法非常简单、基于事件驱动型的编程方法使得很容易上手，因此在之后的一段时间都非常火爆。开源社区出现了大量基于jQuery的相关工具，这样完善生态链更加巩固了jQuery的地位。

然而随着W3C规范逐渐完善，以及Web页面复杂度的直线上涨。前端开发工程师们逐渐厌倦使用jQuery编写面条式的代码，各种各样的前端开发的框架开始渐渐浮出水面，例如：Backbone，React，Ember，AngularJS，Vue等等，其中最为出色的莫过于React，一经开源，就广受开源社区的好评。在这之后，Vue凭借广大的中国开发者强势崛起。AngularJS2也完成发布。主流的JavaScript开发框架也形成了三足鼎立之势（React，Vue，AngularJS）。2008年，随着第一版Chrome发布，V8引擎也正式发布。ECMAScript5也紧随V8之后完成发布。V8引擎发布后，node.js也孕育而生，前端也逐渐扩大到服务端开发。各种构建工具也默默的不停迭代，随着Grunt逐渐没落，Gulp，Webpack相继出现。但明显Webpack更受开源社区的青睐，并在2016年成为了主流的前端构建工具。

从二十世纪的兴起，到现在的短短十几年发展，就已经表现出了强大的生命力，到目前发展来看，前端开发技术的发展时间相对较短，但是发展速度却非常快，并且技术的迭代也十分迅速，这和当前互联网的极速发展有着密不可分的关系。一个稳定、流畅、对用户友好的前端界面往往能够给用户带来非常良好的使用体验，非常大的提升了互联网用户的满意度。文本所研究的基于React、Redux和Webpack的前端优化方案，正是对当前主流前端开发技术的深度整合和性能优化，基于本方案能够在减少开发者开发难度的同时，极大的优化网页的性能和质量，在更小的打包体积下极大提升网站的响应速度和兼容性，获取更好的用户体验。

## 1.2论文主要内容

本文主要研究网站前端优化方案，通过React、Redux和Webpack框架的深度整合和性能优化，在减少开发者开发难度的同时来优化网页的性能和质量，在更小的打包体积下极大提升网站的响应速度和兼容性。论文包括五个部分，每一部分的内容组织如下：第一部分是绪论，主要介绍了论文的选题背景以及研究意义，包括当前前端技术的发展现状以及发展历程，并在最后介绍了全文的内容安排；第二部分是对本文相关概念的进行阐述，并且对本文主要涉及的相关技术进行深入的探讨；第三部分是全文的重要部分，分为三小节，分别阐述了原型的设计与实现，对原型开发和生产环境下的优化方法，以及对原型进行自动化测试和持续集成；第四部分是对第三部分提出的方法的性能分析，主要通过对网站各项性能指标进行对比，来评估此优化方案所带来的性能提升；第五部分是论文的结论，指出文章的研究结果和局限性。

# 

# 2.相关技术分析

## 2.1 React

React是由Facebook于2013年5月开源的JavaScript库，主要用于网页UI的构建。一经开源便饱受好评。React能够如此受到欢迎，和它拥有如下的的特点是密不可分的：

1.高效。通过使用虚拟DOM，能够减少不必要的的交互，从而提高性能。

2.灵活。可以与已知的库和框架很好的配合。

3.组件化。天生的组件化思维，用过组件的方式使代码复用变得十分容易，能够十分容易的构建大型项目。

4.单项数据流。数据自顶上下流动，使得整个组件的状态变得可追踪。

我们知道，如果我们直接操作DOM，那么性能是非常差的，我们可以用图2.1的方法把一个最基础的DOM元素属性都列出来，结果如图2.2，因为代码比较长，这里只粘贴了约1/10。

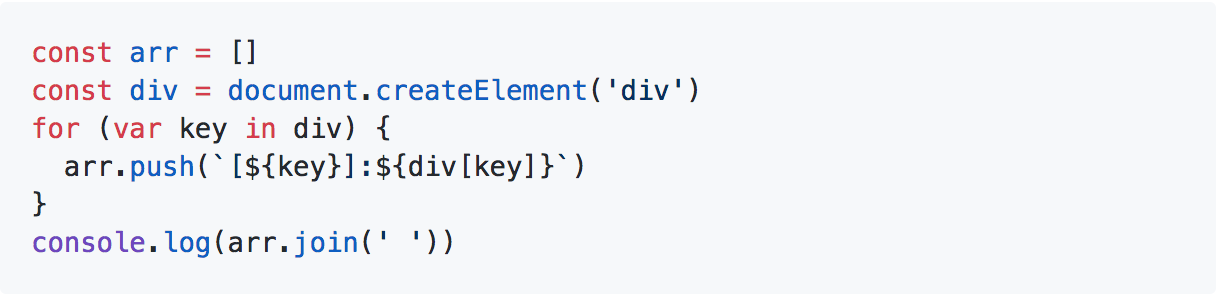


图2.1 获取DOM属性的方法

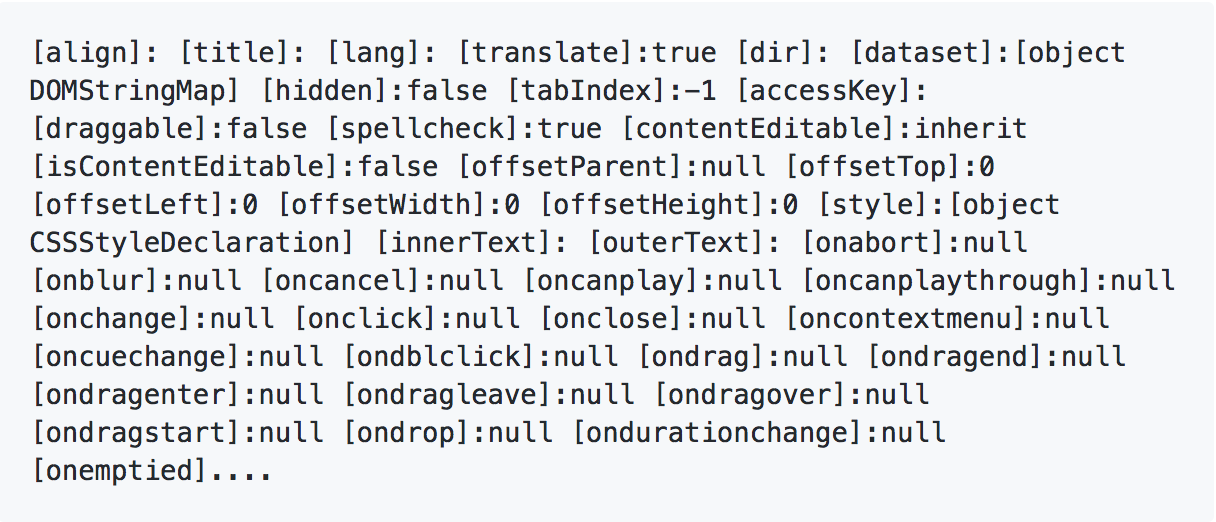


图2.2 第一层DOM元素属性的1/10字段

上面所展示的仅仅是一个空div对象第一层遍历的1/10，真正的对象如果全部展开是非常庞大的。所以我们在操作DOM的时候要非常的小心，不然会导致整个页面重排，这对网站的性能有非常大的影响。

我们可以用一个简单的JavaScript对象来表示DOM节点，这样我们只要操作 JavaScript对象就可以模拟DOM上的操作了，如图2.3所示。



图2.3 用JavaScript对象模拟DOM

可以将上面的JavaScript对象转换成真实的DOM，如图2.4所示。

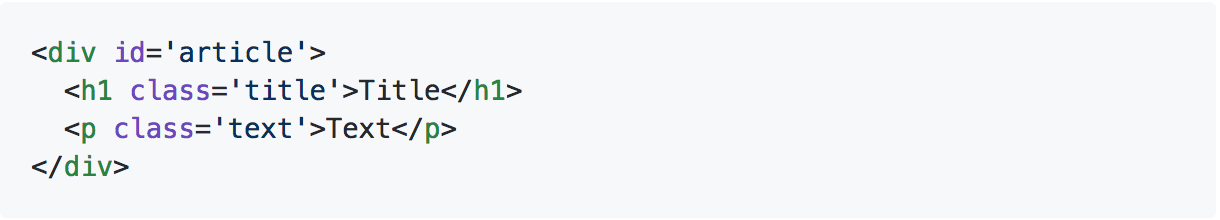


图2.4 JavaScript对象转换为真实DOM

通过这样我们就能在JavaScript对象和真实DOM之间互相转换，就可以用 JavaScript对象表示DOM信息和结构，当状态发生变化的时候，我们可以用新的对象树和旧的对象树进行比较，记录这两棵树的差异。记录下来的不同的就是我们需要最后更新的内容。然后将其应用到真正的DO节点上，页面就更新了。

## 2.2 Redux

React是一个UI库，单靠React不足以搭建一个功能完备web应用。因为在React中数组是从上向下单向的，子节点的数据只能从它的父节点上获取，但是如果节点的嵌套层次非常深，那么如果子节点要获取数据就要通过层层传递来获取，这样就导致了重复的编码工作，也不利于修改。为了管理React的数据，Facebook推出了Flux框架，2015年Redux出现，将Flux和函数式编程的方法合二为一，成为了一时前端框架的热门。Redux的出现就是为了解决复杂场景下的业务逻辑设计的，如果是简单的业务逻辑，可以完全不使用Redux。

想要理解Redux先要理解这四个名称：store、action、reducer、dispatch,如图2.5所示。

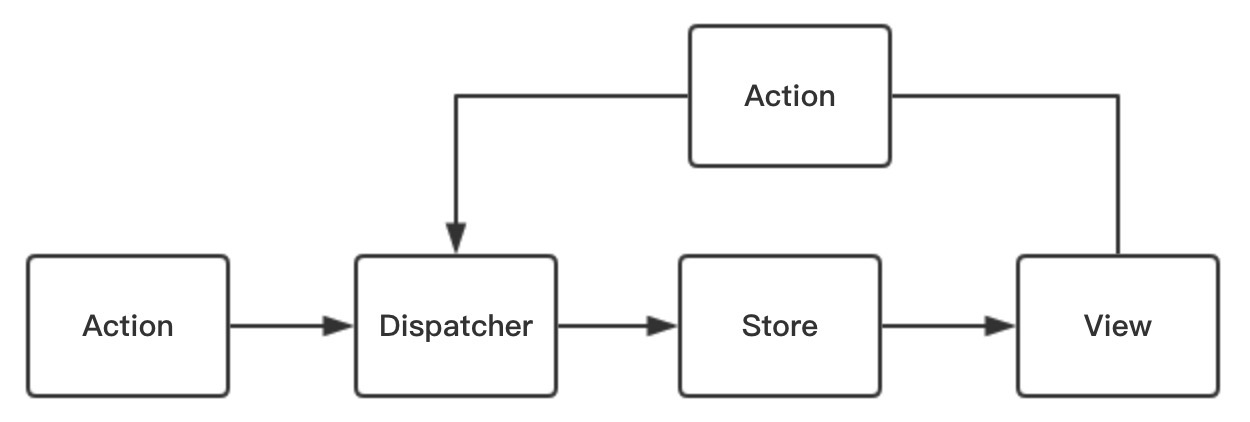


图2.5 Redux各个概念关系图

Store是客户端用来存状态的地方，一个客户端只能有一个单独的Store；Action是用来描述一个特定的客户端事件，可以是输入内容，也可以是点击某个按钮。如果State 发生变化那么View也会相应的变化。但是，用户无法直接改变State，只能操作View。所以，用户想要改变State需要通过一个媒介来完成，那就是Action，用户每一次请求改变状态都会产生一个相应的Action；Reducer是处理Action的场所，它是一个函数，并且接收一个Action作为参数，并且根据当前的State，返回一个新的State；Dispatch是Reducer接收Action用来更新数据的唯一入口。

所以整个数据流就变成这样：整个的享数据报存在Store中，React根据Store和自身的State渲染出View。当用户想更新Store中的数据，则要生成相应的Action，并且回调store流出的dispatch方法，将Action和存在其中的数据传递给Reducer。Reducer接收到Action并且根据当前的State返回一个新的State去更新Store中的数据。从而再由React更新View。

## Webpack

随着前端的发展，一个功能完备的网站所需要的依赖越来越多，几十个上百个依赖是常事。为了整合这些依赖，并且减小文件体积，我们需要用Webpack来简化流程:

1. 模块化，将复杂的程序细化为小的代码片段
2. Scss，Less等CSS预处理语言
3. 通过Babel我们就可以提前使用ES6+的功能，Babel可以将ES6+的代码转换并兼容低版本浏览器。

利用Webpack对项目的代码和依赖进行分析，提取重复模块，并将浏览器无法识别的资源转换为可识别的格式。之后可以经过一系列处理优化，并整合成一个完整JavaScript文件。

在实际使用中，Webpack能做的事情有很多，包括：代码的合并、压缩、混淆、自动生成哈希（优化缓存），前端自动化构建，热更新调试，SASS/LESS和ES6+编译和解释执行，字体文件的打包裁剪压缩等等一些列工作。更凭借其强大的Loaders和Plugins可以让我们完成更加丰富的功能，在必要时也可以自己拓展需要的功能。

# 3.原型设计及实现

## 3.1总体架构设计

本原型采用B/S架构，并且将前后端进行分离，前端采用单页应用技术，通过AJAX请求与服务器进行数据交互。但本文所研究的内容只限于前端单页应用的开发，故不涉及后端服务器部分。

之所以将应用做成单页Web应用，是因为其有如下优点：首先，可以像客户端应用一样快速响应用户的请求、同时具备网页的所有优点；其次，用户体验好，切换界面无加载过程，给用户体验原生应用的感觉；再次，后端不用进行渲染，能够减少对服务器的压力，吞吐能力会提高几倍；最后，良好的前后端分离，后端API通用化。

但同样，单页Web应用也有着如下的缺点：初次加载耗时相对较长；不利于搜索机器人收录网站；书签、导航需要程序实现，复杂度较高；对开发者的能力要求略高。

使用React搭建应用，不仅能够享受单页Web应用带来的所有优点。并且通过React独有的 JSX 语法能够方便快速构建单页Web应用，大大降低单页Web应用的开发难度，节省开发人员的研发和运维成本。React的组件化编程思维，能够达到在一次编写，在多处使用的效果。

但单纯使用React并不能开发一个功能完备的应用。所以我们必须加上一些必要的依赖来解决开发单页Web应用中的其他痛点。例如react-router，它能够让你的单页应用像普通网站一样可以通过不同的URL跳转到不同的界面，可以让你通过简单的 API实现书签，导航等功能。此外还有Redux，它能帮助我们管理所有要在多个组件中共享的数据，并优雅的使用他们。

### 3.1.1原型搭建

首先安装 node.js以及npm， 我这里采用的版本是node.js v8.9.0和npm v5.5.1。之后下载git二进制安装包安装，用来对项目进行版本控制。之后初始化npm配置文件和 git项目，然后安装必要的依赖，包括React、Webpack、Babel、react-router、Redux、eslint、stylelint等，如图3.1所示。

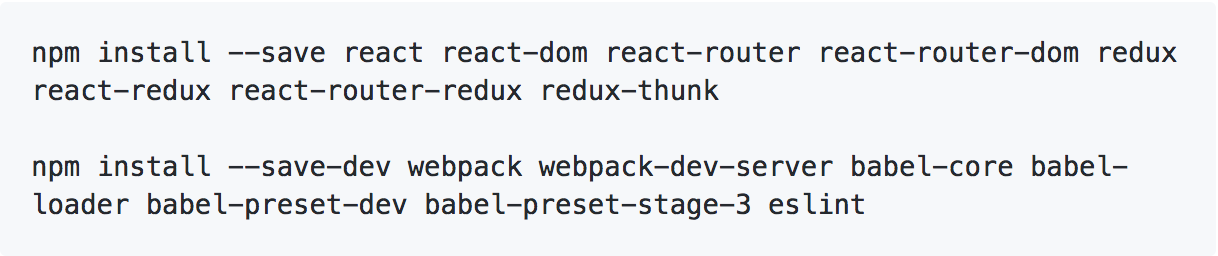


图3.1 安装必要的npm依赖

### 3.1.2用babel兼容最新版JavaScript规范

因为我们所采用的JavaScript是最新的ES2017(ES8)规范，甚至目前主流的 Chrome浏览器还没有完全实现该规范，更不用说还要兼容低版本的IE浏览器。而通过 Babel我们就可以提前使用ES6+的功能，Babel可以将ES6+的代码转换并兼容低版本浏览器。并且因为使用ES6 Modules，故可以将没有使用的代码不打包在最终生成的文件中，在兼容低版本浏览器的同时下还保证了程序不会变的臃肿。并且可以通过添加 babel-preset-react插件来识别React所依赖的JSX语法。通过babel-preset-stage-x插件，可以使用目前未被标准化，但被广泛认同，未来可能会纳入标准的语法。

例如图3.2的ES6+语法：

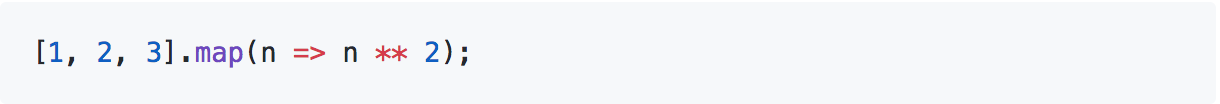


图3.2 使用Babel转换前的ES6+语法

可以转换为图3.3的ES5语法：

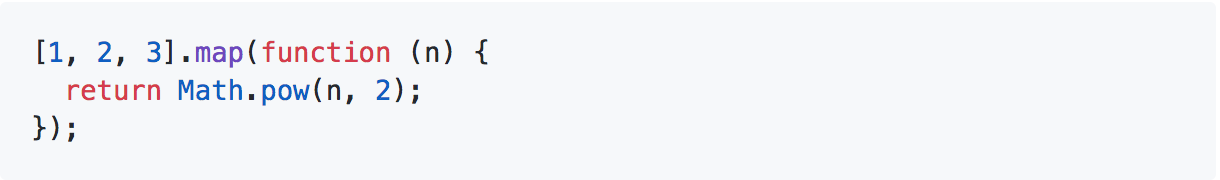


图3.3使用Babel转换后的ES5语法

### 3.1.3用eslint & stylelint对代码静态检查

代码风格和整洁程度也是衡量代码质量的重要指标，毕竟程序是写给人看的，顺便让计算机去执行。如果同一个项目组不能同意一个一样的代码风格，会导致整个代码的可读性非常差。所以，我们的项目也要加入静态检查和统一编码风格的检测。eslint和stylelint就是分别是对JavaScript和CSS做静态检查和统一编码风格。并且都使用当下开源社区都认可的standard配置。

## 3.2原型的优化方案

### 3.2.1 CSS样式优化

为项目添加PostCSS支持，PostCSS是通过Node.js来对CSS进行一系列处理的工具，它的功能十分强大。其中最被我们熟知的是Autoprefixer，它通过配置的要兼容的浏览器版本，自动为CSS添加相应的前缀。

处理前如图3.4所示：



图3.4 PostCSS处理前的代码

处理后如图3.5所示：



图3.5 PostCSS处理后的代码

我们都知道，全局命名空间是一个强大但是非常危险的技术，一旦使用就会给未来留下不易察觉的隐患。如果在写代码的时候，如果你的类名不能保证互不相同的，那么很有可能会导致命名空间冲突，从而导致出现各种各样的问题最终整个代码难以维护。CSS并不支持独立的作用域，但是CSS-Modules的出现填补了这样的空白。CSS-Modules 可以使用node.js处理所有出现的CSS类，并将每个模块单独起一个唯一的类名。然后用JSON记录原始类和当前类的对照关系。这样就解决了CSS的全局污染。可以使开发者更高效的开发，并且对项目重构也更加友好。这样每个单独模块下的CSS都是独立存在，互不依赖，不会因为全局命名空间而导致牵一发而动全身的情况了。

### 3.2.2图标优化

最开始的时候，网站的图标都是用jpg/png格式的图片来展示的，但是随着项目逐渐变大，慢慢的会发现网络请求中的图片资源的请求量占了很大一部分，减少图片资源请求量就变得至关重要了。为了解决这个问题，孕育而生的就是雪碧图。具体做法是将多个图片放在一张图片上，然后通过background-position定位到特定位置的图片来使用单独的图片资源。这样做虽然解决了图片细碎的麻烦，但是却依然存在难以维护的困惑。每次新增图标都要重新生成，既不能利用缓存也不太方便。

之后又出现了使用字体库来实现页面图标的，例如material-design-icons，虽然可以方便使用，但是如果要拓展图标库却很麻烦的。这时是阿里巴巴的图标库Iconfont就出现了，并且可以将自己的图标上传供大家使用。Iconfont主要有三种使用方式，第一种是Unicode、其次是Font-class、最后是Symbol。它们的具体区别如表3.1所示：

**表3.1 Iconfont三种使用方式具体区别**

| 方式 | 兼容性 | 调整大小颜色 | 多色图标 | 使用格式 |
| --- | --- | --- | --- | --- |
| Unicode | ie6+ | √ | × | <i class="iconfont">&#xe604;</i> |
| Font-class | ie6+ | √ | × | <i class="iconfont icon-xxx"></i> |
| Symbol | ie9+ | √ | √ | <svg class="icon" aria-hidden="true"><use xlink:href="#icon-xxx"></use></svg> |

数据来源：Iconfont官方文档

随着一些老旧的浏览器逐渐被淘汰，使得我们不必要再去考虑这些繁杂兼容性问题。

Symbol在内部使用的是SVG图标，因此有如下优点：支持彩色图标，不受单色限制；支持通过CSS来调整样式；可以用CSS实现动画；SVG为矢量图，缩放不会失真；相比于image，整体更小；可以内联到html中，减少http请求。

但是直接是使用Symbol还有一个很大的问题，现在所有的图标都是通过Iconfont引入的。每次增删图标只能将整个iconfont.js文件一起替换。其次也无法做到按需加载，不能根据使用了哪些SVG来动态加载。也不方便将自己做的SVG图标整合到Iconfont上，只能上传然后再重新生成并且下载，十分繁琐。

这时候就需要我们自己制作svg-sprite了，不过为了工程化和自动化，我们可以使用Webpack编写一个loader来自动化整个制作svg-sprite的过程。结合我们的项目，我们将src/icons目录下的所有SVG图标都交给svg-sprite-loader这个 loader 去处理，并且其他目录下的图标不受此影响，如图3.6所示。



图3.6 在Webpack中使用svg-sprite-loader

配合Webpack的require.context函数，我们就可以不用手动引入SVG图标。只需要将图标放到之前定义好的src/icons文件夹下，当需要这个图标时就会自动在这个文件夹中寻找。之后我们将这部分封装成一个React组件，就能在想使用图标的地方这样使用了，如图3.7所示。

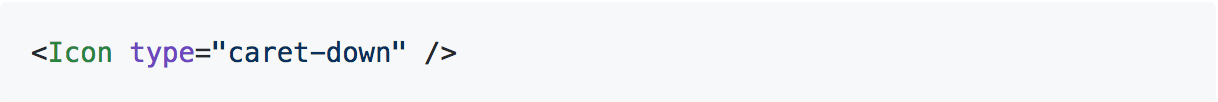


图3.7 使用SVG图标的方法

之后我们还要使用SVGO来去掉SVG里无用的信息，来进一步优化 SVG的大小，我们先看一下没有经过优化的SVG是什么样的，如图3.8所示。

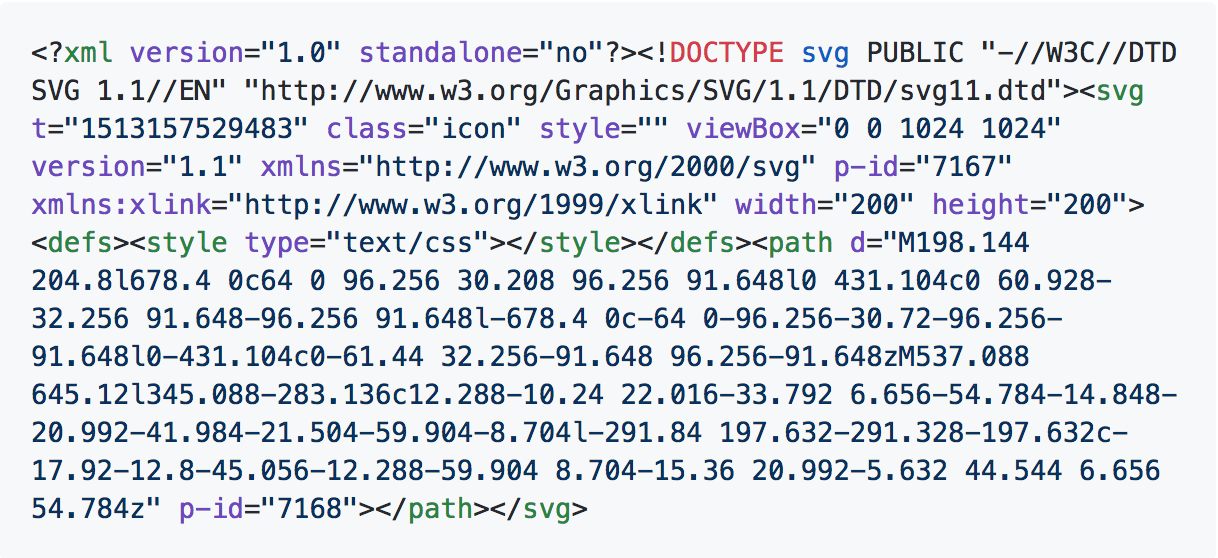


图3.8 使用SVGO优化前的SVG图标

经过SVGO优化后，我们的图标变成了这样，如图3.9所示：

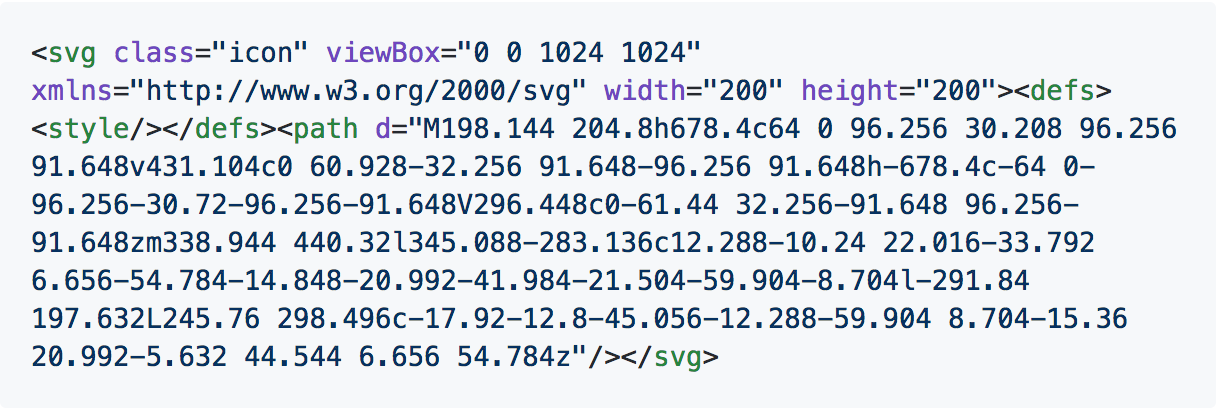


图3.9 使用SVGO优化后的SVG图标

可以明显看出整个文件变小了很多，并且对显示没有任何影响。经过这一系列的优化，我们终于可以优雅的使用SVG。

### 3.2.3 Webpack性能优化

对Webpack进行性能方面的优化其实分为两个方面，第一是开发时候需要进行的优化，其次是生成环境需要进行的优化。

模块热替换：开发网页时，每当写完一个小功能或调整一个样式，总要切换到浏览器，然后刷新浏览器查看效果。这就造成了开发效率的低下，调试的时候大部分时间都浪费在无意义的应用切换和点击刷新按钮。这时候就要使用Webpack提供的模块热替换功能了。热模块替换是Webpack提供的众多功能中最有用的功能之一。其内部是使用 node.js开启一个开发服务器，并监听本地代码文件变化，当代码发生变化的时候会重新编译项目，并找到两次版本不同的部分，通过JSON的形式发送给浏览器中的HMR Runtime。浏览器再解析JSON文件，动态更新修改的部分，就实现了模块热替换。从而当代码发生变化的时候，浏览器也给出相应反馈，大大提高开发效率。

生成source map：我们的源代码是用ES2017(ES8)写的，但是运行在浏览器中的代码是经过Babel编译转换后的，调试的时候就很痛苦了。这时我们就需要source map 来映射编译前前的代码。而且要确保生成source map的时候要足够快，不然编译的过程就太慢了。所以采用了cheap-source-map这种编译速度快，但是不太安全的方法。在正式上限的时候一定不能采用这种方法。

分离第三方库：第三方的代码库一般在我们架构整个项目的时候就会确定使用的库和相应的版本，并且在日后的开发维护中很少改动，所以我们可以将这些变化小的部分独立抽离出来。而且还能够减少热模块替换时的编译速度，使改变更快的应用到浏览器上。通过这种方式，每次打包的时候，都会计算库包含的所有代码，若干库中的文件不变，那么生成的Hash就相同。所以只需要打包一次，只要库文件不变，就可以一直使用。

编译文件分析：一个成熟的项目，需要的npm依赖包也是十分多的。就拿本项目说就已经69个依赖了，如果这当中有写没有用到的被打包进项目中，那岂不是太影响最终的效果了。所以我们需要webpack-bundle-analyze这个插件，来帮我们可视化的分析出项目每个依赖的大小和依赖关系，使我们能更加极致的优化依赖关系。

自动添加hash优化缓存：缓存是浏览器对静态资源加载的一种非常重要的优化方式，但有时我们更新了某个文件或者图片。但是刷新网页并没有改变，这就是缓存的原因了，我们要在CSS或者图片等资源后加上类似于这样"base.css?v=0.3"版本号，但是手动改版本号太过于繁琐，往往会因为程序员疏忽漏掉。而这种活交给Webpack干就非常容易了。Webpack可以在生成每个文件的时候，将文件的名字替换成文件的hash。所以只要两份代码完全相同，那么他们的名字也就相同，起到了缓存优化的作用。

分离打包CSS和JavaScript：Webpack在打包的时候，会把所有的资源都当成模块。包括JavaScript、CSS、图片、字体等资源，都可以打包在一个JavaScript文件中。但是如果我们改变部分JavaScript代码，会导致整个应用的hash发生变化，所以CSS缓存也会失效。但是我们可以将程序中的CSS和JavaScript打包到不同的文件，将业务逻辑从样式中抽离出来，从而更好的利用缓存。

压缩混淆JavaScript：通过UglifyPlugin可以将Webpack编译后代码进行压缩，并且使用更短的变量名代替原来的变量名。使生产环境的代码变得更小，下载源文件所需要的时间大大缩短。经过混淆的代码十分晦涩，能够让你的代码没有那么容易被破解。

## 3.3原型自动化测试及持续集成

项目的维护如果全部靠人来保证，那么这个项目也就存在着风险。毕竟人总有犯错的时候，尤其当项目非常庞大的时候，甚至是项目负责人都无法了解项目的全部代码，那么这个时候，测试的出现就可以很好的弥补这个缺陷。

### 3.3.1自动化测试

一个自动化测试体系一般包括如下四个部分：测试运行器：edp-test karma；测试框架：jasmine mocha qunit Jest；断言库：expect.js should chai；覆盖率：istanbul。在开发软件的时候，我们可以先写好测试，然后逐步满足测试所需的功能。这种开发方式的好处能够非常大程度上减少低级失误，在未来需要对这部分代码重构或者修改的时候，可以将修改代码带来的隐患降到最低。

本原型所采用的是mocha + Istanbul + chai。我们选用mocha的理由是因为mocha不仅可以运行在浏览器同样可以在node上运行，所以我们不需要karma框架作为测试运行器。

### 3.3.2 持续集成

在我们的实际开发中，一个大型项目通常都需要多人配合完成，这就需要软件的开发成员要经常同步他们的代码，也就意味着每天要多次提交他们的代码到代码仓库中。为了不将错误引入项目中，每一次的代码提交都应该通过自动化的编译、测试来进行验证，从而尽可能早的发现问题并解决问题。在我们的原型采用的是 Travis CI。

我们要在根目录下创建 travis.yml 文件，并添加如图3.10的配置：

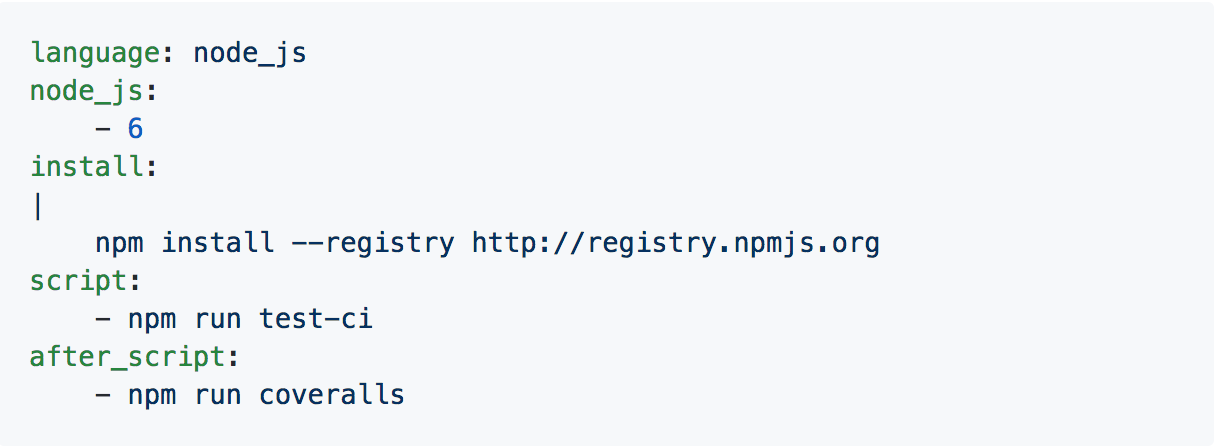


图3.10 travis.yml配置文件

我们需要在配置文件中添加所用的语言、相应版本、如何安装所需依赖、如何执行测试代码。当配置完成后并提交代码时，Travis就会检查是否存在该配置，如果有就运行测试脚本。如果脚本运行中发生故障中断，还会给项目提交者发送一封提醒邮件。提交者可以通过日志查到具体的原因，从而尽早发现并解决问题。

当完成测试时，会调用after\_script中的npm脚本，进行代码覆盖率检查，代码覆盖率报告可以通过一系列的数据可视化的手段将代码测试覆盖的情况体现出来，让用户能够清楚的知道这个软件是否经过系统的测试。

# 优化效果评估

本章是针对对第三章提出的优化方案，我们主要从以下几个性能指标去测试我们的优化效果。

## 4.1 lighthouse测试

lighthouse能够分析网络应用程序和网页，并对比最佳实践给出一个程序的评分。

优化前如图4.1所示：

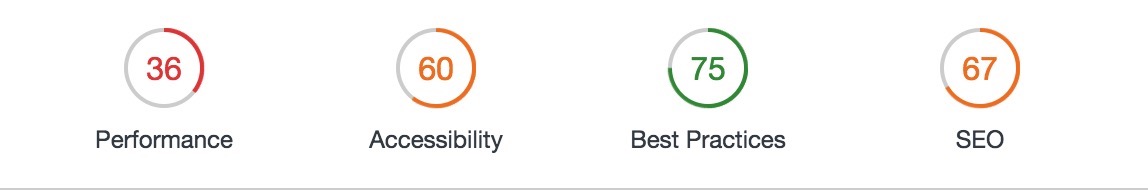


图4.1 优化前lighthouse评分

优化前如图4.2所示：



图4.2 优化后lighthouse评分

首先是性能指标，经过优化从36分提高到了69分，涨幅非常明显；其次是安全性，应为我们的程序还没有对安全性进行进一步的优化，所以这里的评分并没有变化；再后面是针对最佳的评分，从75分提高到了81分，已经非常不错了。最后是SEO，这是SAP技术的共同缺陷，所以分数并没有提升。

## 4.2打包大小

优化之前的打包文件如图4.3所示：

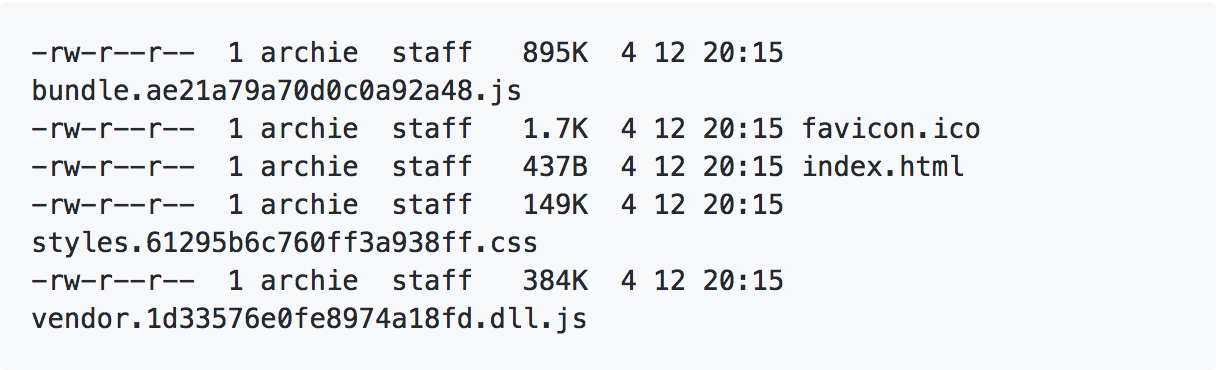


图4.3 优化前打包文件

优化之后的打包文件如图4.4所示：

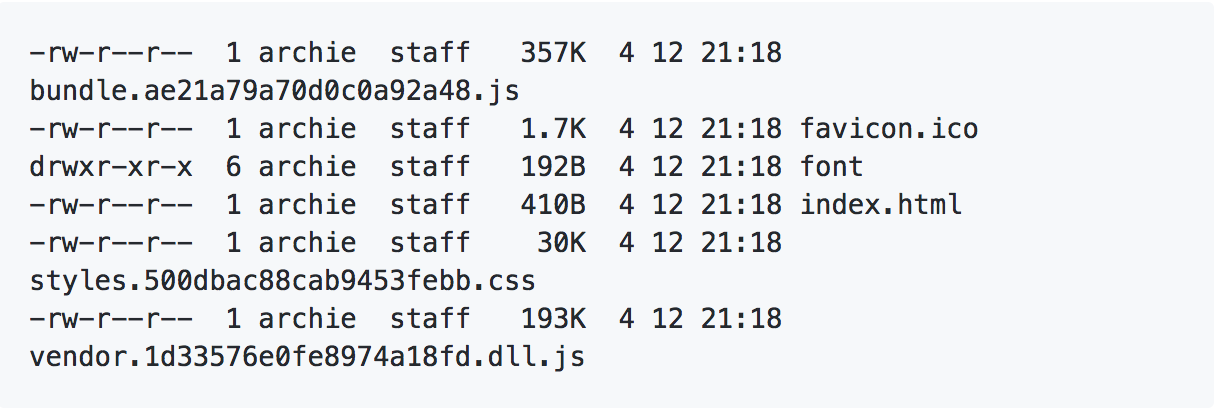


图4.4 优化后打包文件

其中index.html是我们网站的入口文件；bundle.js是程序的业务逻辑代码；vendor.dll.js是和业务逻辑没有关系的第三方代码库，我们将其抽离出来，单独打包成一个文件，这样可以在网站更新的时候复用这部分文件的缓存；styles.css是我们的样式文件；还有font文件夹，里面存放是是经过优化的图标文件，优化之前这些图标是存放在bundle.js中。经过整理后，我们得到的结果如表4.1所示：

**表4.1 优化效果对比表**

|  |  |  |  |
| --- | --- | --- | --- |
| 文件 | 优化前 | 优化后 | 优化效果 |
| index.html | 437B | 410B | 93.82% |
| bundle.js | 895K | 357K+192B | 39.89% |
| dll.js | 384K | 193K | 50.26% |
| Styles.css | 149K | 30K | 20.13% |
| all | 1237K | 580K | 46.89% |

通过上面的图表，可以看到我们的优化效果非常明显，尤其是CSS文件，少了约4/5的大小，整体大小减少了53.11%。打包后的模块图如图4.5所示：

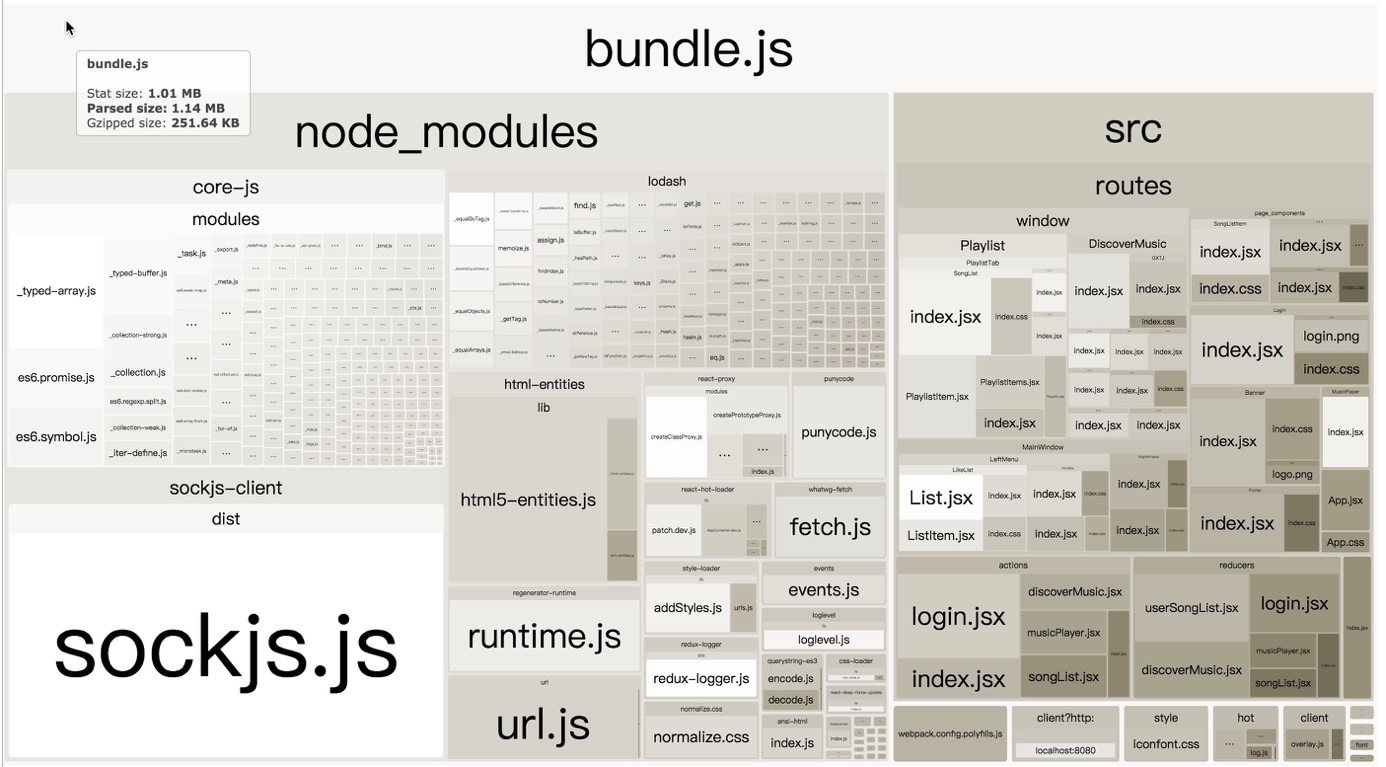


图4.5 Webpack模块打包图

# 结论

通过使用React进行UI构建，然后使用Redux来集中管控数据，然后利Webpack对项目的代码和依赖进行分析，提取重复模块，并将浏览器无法识别的资源转换为可识别的格式。通过babel将ES6+的代码转换并兼容低版本浏览器；PostCSS对CSS进行兼容和拓展，并用CSS-Modules来对CSS进行作用域隔离。使用Iconfont对图标从大小到兼容性进行优化。并在此基础上添加了自动化测试以及持续集成。

经过我们的理论研究和原型测试，我们发现上述的优化方案确实非常有效果。我们可以使用CSS next和ES6+的语法进行编程，同时也可以获得非常好的兼容性，最终经过打包后的程序可以减少50%以上的大小，网络请求的数量可以从几十个下降到几个，整个的网站的响应速度可以有3-4秒的提升。并且用户在网站内进行交互时，可以得到客户端一样的使用体验。整个优化不仅可以在减少开发者开发难度，并且可以同时优化网页的性能和质量，在更小的打包体积下极大提升网站的响应速度和兼容性。

# 参考文献

[1][A JavaScript library for building user interfaces.](https://reactjs.org/) 2018

[2][Declarative routing for React](https://reacttraining.com/react-router/) 2018

[2][Redux is a predictable state container for JavaScript apps.](http://redux.js.org/) 2018

[3][webpack is a module bundler for modern JavaScript applications.](https://webpack.js.org/) 2018

[4][The compiler for writing next generation JavaScript](http://babeljs.io/) 2018