Classified Index:
U. D.C:

Southwest University

of Science and Technology

Master Degree Thesis

Research on Performance Optimization Framework Model based on Web Front-end

Grade: 2015
Candidate: Du Yan-mei
Academic Degree Applied for: Master
Speciality: Computer Science and Technology
Supervisor: HUANG Xiao-fang

Nov.06,2018

摘 要

随着 web 应用功能的丰富化，web 终端的多元化发展，如何快速迭代产品以满足市场需求，提升企业竞争力，对于前端技术的发展提出了更高要求。传统的 B/S（Browser/Server，浏览器/服务器）模式下的 web 开发中存在前端代码无法复用；移动端性能表现不佳；单页 web 应用不能满足用户体验需求；最重要的是，传统的开发模式下，网页路由需要后端配合，导致前后端工作交叉等待，产品迭代和维护效率较低，无法满足企业级应用的迭代速度。因此，如何针对企业级的 web 应用，研究如何优化前后端分离开发技术，提出一种解决多终端性能、组件化开发和打包部署的完整的开发模型，并进行实践应用证明，对于企业级应用的开发具有重要的意义和研究价值。

本文通过研究前后端分离模型，研究前后端分离的开发模式，提出一种高效的前后端技术分离模型，设计了面向 SaaS 应用的前后端分离模型，并提出了面向企业业务应用场景的组件化开发技术框架，实现了前后端的并行开发，减少了前后端开发的耦合性。为了减少后台访问压力，本文在基于 Nignx 的负载均衡算法基础上，实现 web 前端性能的优化，大量的静态页面的访问可以通过 Nignx，后台不再关心静态页面访问带来的压力，提升了 web 网站的处理效率。最后，本文通过企业级开发应用的实践，将上述模型及方法应用在一款上线的企业级 web 产品开发中，通过实践证明该开发模型能够满足多终端的设计及运行需求，在访问及使用过程中，效果较好，能够满足企业级 web 应用程序的大型分布式架构、弹性计算架构、微服务架构、多端化服务的开发需求。

关键字：前后端分离；组件化开发；Node.js；Vue

Abstract

With the enrichment of the web application function and the diversified development of Web terminal, how to quickly iterate the product to meet the market demand and improve the competitiveness of the enterprise has put forward higher requirements for the development of the front-end technology.In the traditional B/S (Browser/Server) mode, the front code can not be reused in the development of web; the performance of the mobile terminal is not good; the single page web application can not meet the user experience needs; the most important thing is that under the traditional development mode, the web routing needs back end coordination, leading to the cross end work crossing, etc. The efficiency of product iteration and maintenance is low, which can not meet the iterative speed of enterprise application.Therefore, how to optimize the front and back separation and development technology for the enterprise level Web application, and put forward a complete development model to solve the multi terminal performance, component development and packaging deployment, and carry out practical application to prove that the development of enterprise application is of great significance and research value.

By studying the model of front and back separation, this paper studies the development mode of front and back separation, proposes an efficient separation model of front and back end technology, designs a front and back separation model for SaaS application, and puts forward a component development frame for enterprise business application scene, and realizes the parallel development of front and back end. Less coupling of front and back development.In order to reduce the pressure of background access, this paper optimizes the performance of Web front-end on the basis of the load balancing algorithm based on Nignx. A large number of static page access can be accessed through Nignx, and the pressure of static page access is no longer concerned in the background, and the processing efficiency of the web web site is improved.Finally, through the practice of enterprise development and application, this paper applies the above model and method to an on-line enterprise level web product development. Through practice, it is proved that the development model can meet the design and operation requirements of multi terminal. In the process of accessing and using, the model can meet the enterprise level web application. Large scale distributed architecture, flexible computing architecture, micro service architecture, and multi terminal service development needs.

Keywords: Separation; Component-based development; Node.js; Vue

目 录

1 绪论

本章首先介绍了web前端性能优化的课题研究背景和意义，然后对web前端性能优化国内外的研究现状进行了详细的描述，最后介绍了本文的研究内容和论文结构设计。

1.1 课题研究背景与意义

宽带的普及,让网速变得越来越快的同时也让网页越来越复杂,用户的耐心越来越少，网页打开时间稍长，就会造成大量的用户流失。对于Web性能的追求不仅仅是用户体验上的需要，更是互联网公司核心业务的驱动力。无论是靠电商、游戏、娱乐、社交、搜索引擎功成名就的IT巨头，都在实践中证明：网站越快访问体验越好，用户黏性越高、用户忠诚度更高、用户转化率也越高。可见，随着互联网的发展，web应用的复杂化，web性能优化的重要性有增无减。

随着前端技术的飞速发展和web应用的复杂化，web应用正在往兼容多终端、高性能的方向发展，为了快速迭代产品以满足用户日益变化的需求、提高用户体验及提升企业竞争力，web应用开发技术要求更加高效率的开发，并兼顾性能要求，方便快速迭代及后期维护简单。而传统的B/S（Browser/Server，浏览器/服务器）模式下的web开发中存在前端代码无法复用；性能在移动互联网中很难达到最优；大量多页面应用，单页web应用(SPA，single page web application)不能满足需求；而且，在这种开发模式下，网页路由需要后端配合，前后端工作无法独立，
再次迭代和维护效率较低，无法满足企业级应用的迭代速度。因此，随着前端技术的飞速发展，如何通过对web研发模式进行改进，使得前端和后端实现完全解耦，具有重要的研究意义和使用价值。

本文深入研究前后端分离的开发模式，提出一种解决多终端兼容，解决web性能的组件化开发和打包部署的开发模型，提升前后端开发效率，最后通过实际系统的开发，证明该模型的高效性。

1.2 国内外研究现状

2010年之后，Web2.0深入人心，W3C宣布成立Web性能工作组，Mozilla

以及Google纷纷推出应用商店，浏览器调试工具丰富了起来，人们开始更多地关注开发体验和性能问题。

雅虎公司作为互联网公司的先驱，其前端团队提出14条网站性能优化原则[1][2]：尽量减少HTTP的请求数，使用CDN，添加Expires头，启用Gzip压缩，将CSS放在页面最上面，将script放在页面最下面，避免在CSS中使用Expressions，把JavaScript和CSS都放到外部文件中，减少DNS查询，压缩JavaScript和CSS，避免重定向，移除重复的脚本，配置实体标签，使AJAX缓存。其给出的优化建议多是基于现行HTTP协议展开的，并未对协议进行改进。

Google公司对Web应用程序前端的性能优化也非常重视，其研究和开发了多款提升前端性能的应用，包括：

(1) Web前端性能的分析插件Page Speed；

(2) Javascript和CSS的压缩工具Google Closure Compiler；

(3) 自主研发的V8 JavaScript引擎在运行之前将JavaScript编译成了机器码，以此提升性能；

(4) google的前端框架AngularJS实现构建单页面应用，得到广泛使用，但是，伴随着模块化在ES2015的尘埃落定,更多优秀的框架如reactJs、VueJs,对AngularJS的首要地位造成撼动,其中重要的一点是AngularJS文件太大，对性能有一定的影响，因此2015年新出的AngularJS2.0在性能方面进行一定的改进，提升其竞争力；

(5) Google甚至在协议上进行改进，Google2012年开始SPDY项目，致力于在HTTP/1.x之上实现更快的速度，目前处于起草阶段的HTTP2也是基于SPDY的标准，HTTP2的出现让web性能得到大幅度的提升，目前已得到众多大公司的应用和浏览器（其中就有google Chrome）的支持，但是HTTP2的普及还需要相当长的时间，google为了推广HTTP2，宣布在2016年停止对SPDY项目的支持。

由上可见，互联网巨头google公司对于web性能也是非常重视的。

相比国外，国内的web前端研究起步较晚，但是国内web前端技术发展迅猛,近几年国内领先的IT公司在web前端性能方面做了大量的研究和实践。

国内大企业诸如淘宝、腾讯、新浪、百度、搜狐等都对自己的网站进行了重构，从代码级别和页面级别提升网站的性能，为了更准确且持续性地对网站性能进行优化，百度搭建了自己的性能监控系统持续监控、评估、预警页面性能状况、发现瓶颈，指导优化工作的进行。淘宝前端团队自主开发高

性能的库 KISSY，采用模块化思想，从架构方面去解决淘宝 web 应用的性能问题，百度开发的前端构建工具 FIS3 旨在解决前端开发中性能优化、模块化框架等问题。国内中小型企业对 web 前端性能的优化也产生了重视，分别从不同程度去进行 web 应用的性能优化，以期提升网站访问速度和竞争力。

在学术界，针对 web 端性能的优化，也有大量的研究工作。2016 年，孔令旭等人提出了一种基于 Node.js 实现的前后端分离的解决方案，并证明了其在性能方面的高效性[3]；2017 年，吴贺提出一种前后端解耦模式的开发方案，其核心是通过在传统的 MVC 结构上，引入 Node.js 作为前端和后端的中间层，从而解耦前端和后端，实现前端独立开发[4]；同样在 2017 年，仇晶、黄岩等人提出基于 Node.js 中间层的 web 开发，在传统的互联网前后端 web 应用中，提出了引入 Node.js 作为中间层来解除前、后端之间的耦合关系的新解决方案[5]。

但是，我国学术界在针对企业级大型 web 应用前后端分离，如何提升前端开发效率等方面研究较少。目前,大部分的系统架构如图 1-1 所示，虽然有些系统采用分布式架构，层与层之间使用了远程调用框架，但是本质类似。前后端耦合的问题主要发生在控制层（Control），控制层是前端和服务端交互的边界，但是在开发过程中控制层（Control）和服务层（Server）常常混淆不清，这就是前后端耦合度高的重要原因。

图 1-1 目前大部分的系统架构

Fig.1-1 Most of the current system architecture

因此要前后端解耦，就是要划清控制层的边界。同时，前后端分离的终

极目标应该是前端和服务端是完全独立的项目，前端项目应该包含图1里的浏览器和控制层，服务端项目包括服务层、DAO层等等。项目开发时候让前端脱离于后端，让后端专注于业务服务，最后生产发布也要独立部署，前端专注于展示和交互，这样就达到了前后端真正解耦。

1.3 研究内容

本文通过研究前后端分离模型，设计面向企业业务应用场景的组件化开发技术框架，并基于Nignx的负载均衡算法，实现web前端性能的优化，静态页面的访问可以通过Nignx，后台不再关心静态页面访问带来的压力，提升前后端开发效率。最后，通过在企业级开发中进行应用，实践证明该开发模型能够满足多终端的设计及运行需求，并具备高效性。论文结果为企业级web应用程序的大型分布式架构、弹性计算架构、微服务架构、多端化服务打下坚实的基础。总结本文的研究内容主要包括如下几方面：

(1) 研究前后端分离的开发模式，提出一种高效的前后端技术分离模型，设计了面向SaaS应用的前后端分离模型的设计与开发，并设计了面向企业业务场景的组件化开发模型。

(2) 通过实际系统的开发实践，证明该模型在SaaS应用开发过程中的高效性。

(3) 对SaaS应用进行了业务介绍和详细的概要设计。将本文设计的框架模型应用到系统中，包括前后端分离模型、组件化开发方案和负载均衡方案在系统中的应用，大大提高了应用的开发效率和页面的响应速度。

(4) 对基于web前端性能优化模型的SaaS应用进行了性能测试和对比分析，得出采用前后端开发模式使得开发效率大大提高，并且性能显著提升，且并发请求量越高，新模型的性能优势越明显，页面的响应速度提升明显。

1.4 论文结构

本论文分为六章，对web前端性能优化模型以及模型的验证和实践进行了详细的阐述，各章节内容如下所示：

第一章主要介绍了本课题的研究背景与意义，详细阐述了国内外研究现状，并总结了本文研究内容，最后介绍了本文的结构。

第二章主要针对本文提出的web前端性能优化模型的相关技术进行分析。

首先对 Json 技术做了介绍，然后详细介绍了 Node.js 技术以及以及基于 Node.js 的应用。然后从组件化开发、前端 MVC 的构建模型和主流框架技术对前端框架技术做了详细的阐述,并介绍了自动化构建工具 webPack 的功能。并对负载均衡算法进行介绍，探讨其用于前端性能优化的可能性。

第三章对基于 web 前端的性能优化的关键技术做了详细描述。首先针对前后端分离框架模型提出总体设计框架，然后详细阐述面向企业应用场景的组件化开发并结合负载均衡算法，进行性能优化，提出高效的前后端独立开发模式。

第四章主要介绍了 web 前端性能优化模型在 SaaS 应用中的实践。首先对 SaaS 应用的业务进行了介绍和概要设计，然后详细描述前后端分离模型在应用中的使用，并将组件化开发方案引入前端开发中，详细阐述了负载均衡方案在系统中的应用,最后验证了 web 性能优化模型在 SaaS 应用中的可行性和高效性。

第五章对 web 性能优化模型在 SaaS 应用中做了性能测试和对比分析。对模型优化前后，对开发效率、页面响应速度做了对比分析，得出相比传统的前后端 MVC 开发模式，采用前后端分离的开发模式使得开发效率大大提高，并且性能显著提升，且并发请求量越高，新模型的性能优势越明显，页面的响应速度提升明显。

第六章是总结与展望。对本文的内容做了全面的总结，对下一步的工作重点做了展望。

2 相关技术分析

本章介绍了基于 web 前端性能优化模型的相关技术，包括 Json、Node.js 以及组件化开发思想、基于前端 MVC 的构建模型和主流前端框架技术，并介绍了自动化构建工具 webpack 和负载均衡组件。

2.1 Json 技术介绍

JSON 是一种 JavaScript 对象表示法（JavaScript Object Notation），是一种轻量级的文本数据交换格式。JSON 基于 ECMAScript 的一个子集，采用完全独立于编程语言的文本格式来存储和表示数据。相较于 XML，JSON 以简单的数据结构表示复杂的数据结构，从而简化工作量。虽然 JSON 使用 Javascript 语法来描述数据对象，但是 JSON 独立于语言与平台，针对 JSON，许多编程语言都有解析器和序列化器。JSON 简洁和清晰的层次结构让它成为理想的数据交换语言，易于人们阅读和编写，同时也易于机器解析和生成，并有效地提升网络传输效率。在语法上，JSON 文本格式与创建 Javascript 对象的代码相同。在 Javascript 的语法中，一切皆对象，因此可以用 JSON 来表示任何 JavaScript 支持的类型，比如常见字符串、数字、数组、对象等类型的值。在实际应用中，JSON 常用来表示复杂的数据结构，其中数组和对象是比较常用的两种类型。

JSON 表示对象：首先了解对象在 JavaScript 中是使用花括号包裹{}起来的内容，数据结构为{key1：value1, key2：value2, ...}的键值对结构。在面向对象的语言中，key 为对象的属性，value 为对应的值。键名可以使用整数和字符串来表示，值的类型可以是任意类型。JSON 键值对是用来保存 JavaScript 对象的一种方式，和 JavaScript 对象的写法只有小部分的差异，键/值对组合中的键名写在前面并用双引号（符号为""）包裹，使用冒号（符号为:）分隔，然后紧接着值，如下举例表示 JSON 表示对象和 JavaScript 语法的区别于联系：

```
{"author":  "Jack"}  //这是 JSON 字符串

{author:  "Jack"}  //这是 Javascript 字符串
```

JSON 表示数组：首先了解数组在 JavaScript 中是方括号[]包裹起来的内容，数据结构为[″ Jack″, ″ Joe″, ″ DarLin″]的索引结构。在 JavaScript 中，数组是一种比较特殊的数据类型，它也可以像对象那样使用键值对，但还是索引使用得多。同样，值的类型可以是任意类型。JSON 表示数组和 JavaScript 表示数组一样，都是使用[]表示,如下示例：

```
[" Jack", " Joe", " DarLin"]
```

在实际应用中，前后端数据是比较复杂的，如下使用 JSON 表示一组复杂结构（对象中混合数组）的对象数据：

```
{"meta": {"success": true, "message": "ok"},"authors": ["Jack", "Joe", "Darlin"]}
```

2.2 Node.js 技术介绍

Node.js 是一个采用 C++程序编写和基于 Chrome V8 引擎的 JavaScript 运行环境。Node.js 使用了一个事件驱动、非阻塞式 I/O 的模型，使其轻量又高效，且其可以跨平台使用。Node.js 的包管理器 npm，代替 Node.js 解决了许多部署上的问题，提高开发效率，Node.js 8 已经发布了，NPM 模块每周下载量超过 10 亿，是全球最大的开源库生态系统。

Node.js 逐渐发展成一个成熟的开发平台，近几年，Node.js 吸引了越来越多的关注，其发展和被关注度已然有超越 Java 的趋势。基于 Node.js 的应用层出不穷，国内外诸多大型高流量网站都采用 Node.js 进行开发，比如：PayPal 为全世界 2 亿活跃用户提供服务，PayPal 选择 Node.js 替换 Java 开发后端,使前后端开发只需要使用 javascrip 一种语言,数据统计显示 PayPal 采用 Node.js 之后，应用开发速度提高 2 倍、文件数目减少 40%、代码量减少 33%，并且，接口的请求时间减少了 35%、每秒处理的请求数增加了 2 倍；Mozilla 使用 Node.js 开发了大量应用，统一前后端使用 javascript 一种开发语言，从而提高开发效率；淘宝双十一，天猫首页、天猫的活动页和双十一会场等页面全部都基于 Node.js 应用提供服务，据淘宝开发团队描述，Node.js 不仅帮助前端开发团队高效解决双十一页面渲染上的问题，并且降低了硬件成本；微软、腾讯等也采用 Node.js 做网络应用。

Node.js 的发展，为 web 研发模式带来了新的解决思路，即通过在传统的前端和后端之间引入 Node.js 作为中间层，使得前端和后端实现完全解耦，

业界诸如阿里、百度都已有基于 Node.js 做前后端分离的应用 。

Express 框架是 Node.js 在实际应用中必不可少的一部分，Express[6]是一个基于 Node.js 平台的灵活且极简的 web 应用开发框架，提供一系列强大的特性帮助开发人员创建各种 web 应用，但是不对 Node.js 已有的特性进行二次抽象，它只是在 Node.js 的基础上扩展 web 应用所需的基本功能。Express 提供丰富的 HTTP 快捷方法和任意排列组合的 Connect 中间件，可以帮助开发人员快速创建健壮且友好的 API。Express 框架的核心特性提供：支持通过设置中间件来响应 HTTP 请求；支持定义的路由表来执行不同的 HTTP 请求动作；支持通过向模板传递参数来动态渲染 HTML 页面。Express 的安装和实施也比较简单，其官方文档[6]提供了完整的实施教程。

2.3 前端框架技术

2.3.1 组件化开发

前端作为一种 GUI（Graphical User Interface，简称 GUI，又称图形用户接口）软件，光有 JavaScript/CSS 的模块化还不够，对于 UI（User Interface，即用户界面）组件的分治也有着同样迫切的需求，组件化能大幅提升前端开发效率。随着前端开发复杂度的日益提升，组件化开发应运而生，Web Components 标准的出现更进一步推进组件化“分而治之”的思想，长久地影响和促进前端开发模式，它一方面提高了开发效率，另一方面降低了维护成本，如今应用在移动端占主导地位，各大企业的技术人员研究和期望以 Web 的方式去实现移动应用，多终端复用从组件化的研究逐渐开始。现如今，组件化的 UI 框架层出不穷，比如 AngularJs、ReactJs、VueJs 等前端框架即是对前端组件化开发探索的产物，国内互联网大公司诸如百度、淘宝等前端团队在组件化开发方向上进行大量的研究并投入应用，目前国内大多数企业的前端开发团队正处于组件化开发的探索阶段。

2.3.2 基于前端 MVC 的构建模型

随着 web 应用的复杂化，类似 Jquery 的类库已不能满足复杂页面的开发，为了提高页面渲染性能和开发效率，Web 应用开发理念向越来越强调前端架构的方向发展。

目前，前端常采用单页面技术，即在浏览器端构建 MVC（Model View

Controller，即模型-视图-控制器）模型，这里的 View（视图）已经转为客户端渲染了，通常会使用一些客户端的 HTML 模版去实现，而 Model（模型）和 Controller（控制器），也相应地在浏览器端形成了,这种层面的主流框架有 AngularJs2.0、ReactJs、VueJs。其模型图如下图 2-1 所示：

图 2-1 单页面结构的前端 MVC 模型

Fig.2-1 The front end MVC model of single page structure

2.3.3 主流框架技术

前端框架很多，但是随着前端技术的飞速发展，真正面向前端未来发展的框架较少，但是目前出现的 AngularJs、ReactJs、VueJs 等前端框架是具有未来发展潜力，实现了前端组件化。比如，AngularJs2.0 是 Google 公司的开源框架，其首创的双向数据绑定、内置的模块注入和对组件化的支持，为 web 的发展提供了较大贡献；ReactJs 是 FaceBook 开发的开源框架，是第一个采用虚拟节点（virtual dom）技术的框架，框架足够轻量和灵活，推进了 web 性能的提升；VueJs 是近两年兴起的，具有轻量级、高性能、灵活性的特点，使其不仅适用于大型复杂的 web 应用，而且适合移动场景，为 web 技术的发展和性能提升做了进一步的贡献。

VueJs 是在 2014 年 2 月开源的一个前端开发库，是一套用于构建用户界面的渐进式框架，通过简洁的 API 提供高效的数据绑定和灵活的组件系统，专注于实现数据绑定，模块化与组件化。其核心思想是“数据驱动的组件系统”，本质是基于 MVVM（Model-View-ViewModel）模型的 web 库，通过双向数据绑定连接 View（视图）和 Model（模型）层。实际的 DOM（Document Object

Model,文档对象模型,简称 DOM)操作被封装成 Directives(指令)和 Filters（过滤器）。VueJs 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合，被越来越多的开发团队选择作为前端主要的底层开发框架。在大型的应用中，为了分工、复用和可维护性，我们不可避免地需要将应用抽象为多个相对独立的模块。在传统的开发模式中，我们只有在考虑复用时才会将某一部分做成组件；但在 VueJs 的核心思想中，应用类 UI 可以看成是全部由组件树构成，兼容了组件化的设计思想，是真正面向未来发展的前端框架，框架如图 2-2 所示：

图 2-2 一个 UI 界面拆分成不同组件的关系图

Fig.2-2 A relationship diagram of a UI interface split into different components

2.4 自动化构建工具

分而治之是软件工程中的重要思想，是复杂系统开发和维护的基石，这点放在前端开发中同样适用，目前，前端技术主要采用了模块化方法作为分治手段提高前端的维护效率。

前端模块化包括 javascript 和 css 等的模块化，其中，常用的模块化方案包括 AMD（RequireJs 在推广中对模块定义的规范化产出）和 CMD（SealJs 在推广中对模块化定义的规范化产出）,但是随着 ES6（新版本 javascript 语言的标准）的出现，推进 javascript 的编程趋于标准化和统一化，建立在 ES5 上输出的 AMD 和 CMD 两种不同的模块化方案，现在也可以统一，即采用 WebPack 方案实现前端自动化模块化,给前端性能优化工作带来更多便利。

Webpack 是由 Tobias Koppers 开发的一个开源前端模块构建工具，是一个现代化 javascript 应用程序的静态模块打包器（称为 Module Bundler）。Webpack 处理应用程序的工作原理是通过递归地构建一个依赖关系图（包括应用程序中需要的每个模块），最后将所有模块打包成一个或多个 bundle 文件。Webpack 是高度可配置的，其核心配置中包括入口、输出、loader、插件 4 部分概念：

(1) 入口（entry）:入口（entry）为 webpack 指明了应该使用哪个模块来作为构建其内部依赖图的开始。实现方式即通过在 webpack 配置文件中配置 entry 属性，指定一个或多个入口，具体配置代码如下示例：

```
module.exports = {
        entry: './path/to/myproject/entry/file.js'
    };
```

(2) 输出（output）:输出（output）为 webpack 指明了在哪里输出它所创建的和如何命名 bundles 文件。同样，实现方式即通过在 webpack 配置文件中配置 output 属性，配置具体处理过程，具体配置代码如下所示：

```
cost path = require('path');
module.exports = {
        ......
        output: {
            path: path.resolve(_dirname, 'dist'),
            filename: 'test-webpack.bundle.js'
        }
    };
```

(3) Loader:loader 让 webpack 可以去处理非 javascript 的文件。loader 可以将所有类型的文件转换为 webpack 能够处理的有效模块(webpack 只理解 javascript)，实质上，webpack loader 将所有类型的文件，转换为应用程序的依赖图（和最终的 bundle）可以直接引用的模块。实现方式即通过在 webpack 配置文件中配置 test 属性和 use 属性，具体配置代码如下所示：

```
cost path = require('path');
module.config = {
```

```
    ......
    module: {
        rules: [
            { test: /\.txt$/, use:'raw-loader'}
        ]
    }
};
```

(4)插件（plugins）:loaderd 的作用是转换某些类型的模块，而插件（plugins）功能更加强大，可用于处理范围更广的任务，具体包括从打包优化和压缩，一直到重新定义环境中的变量。实现方式即通过在 webpack 配置文件中配置 plugins 属性和 use 属性，具体配置代码如下所示：

```
cost webpack = require('webpack');   //表示访问内置插件
cost path = require('path');
module.config = {
    ......
    plugins: [
        new webpack.optimize.UglifyJsPlugins(),
        new HtmlWebpackPlugins({template: './src/index.html'})
    ]};
```


Webpack 的基本功能是将以模块格式书写的多个 JavaScript 文件打包成一个文件，同时支持 CommonJS 和 AMD 格式。但让它与众不同的是，它提供了强大的 loader API 来定义对不同文件格式的预处理逻辑，从而让我们可以将 CSS、模板，甚至是自定义的文件格式当做 JavaScript 模块来使用。Webpack 基于 loader 还可以实现大量高级功能，比如自动分块打包并按需加载、对图片资源引用的自动定位、根据图片大小决定是否用 base64 内联、开发时的模块热替换等等，可以说是目前前端构建领域最有竞争力的解决方案之一。其功能架构如下图 2-3[7]所示：

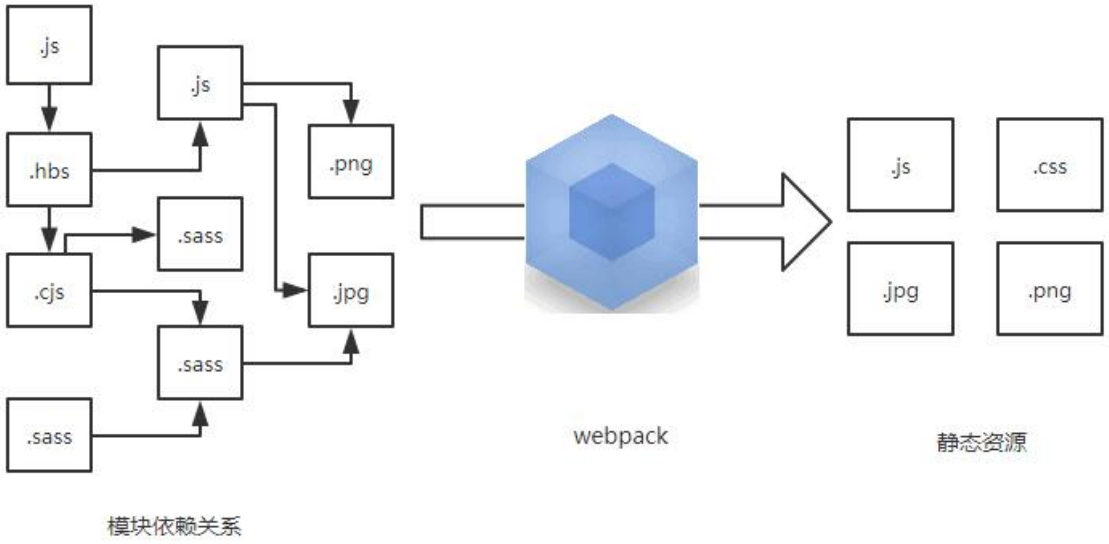

图 2-3 webpack 功能架构图

Fig.2-3 The functional architecture diagram of the webpack

2.5 负载均衡组件

负载均衡（Load Balance）是一种计算机网络技术，是由多台服务器以对称的方式组成一个服务器集群，其将请求均匀地分配到每台服务器，实现每台服务器对外提供服务时承担相同的压力，从而解决并发的问题并提高了业务处理能力。目前，负载均衡常用的应用场景是服务器负载均衡和链路负载均衡，本文的应用场景是服务器负载均衡。服务器负载均衡根据报文层次，目前主流的处理方式分为四层负载均衡和七层负载均衡。

四层负载均衡通过报文中的目标地址、端口以及负载均衡设备设置的服务器选择方式，决定选择真正的目标服务器。七层负载均衡又称为内容交换，通过报文中真正有意义的应用层内容以及负载均衡设备设置的服务器选择方式，决定选择真正的目标服务器。相较于四层负载均衡，七层负载均衡的优点是可以实现更智能的效果，比如将图片和文字类的请求分别转发给图片服务器和文字服务器做处理；另外则是更好的防止 SYN Flood 攻击,给网站提供了更可靠的安全保障。同时，七层负载均衡对设备的性能消耗大，因此对设备的要求更高。目前四层负载均衡主要用于 TCP/UDP 协议的应用，常用在 C/S（全称为 Client/Server，即客户/服务器）模式的系统，而七层负载均衡主要用于 HTTP 协议的应用,常用在网站或者基于 B/S(全称为 Browser/Server，即浏览器/服务器）模式的系统。目前使用最广范的负载均衡是 Nginx、LVS、

HAProxy。

Nginx 属于七层负载均衡，是一款轻量级的 Web 服务器，同时也是一个高性能的反向代理服务器以及电子邮件（IMAP/POP3/SMTP）代理服务器，其具有占有内存少且并发能力强的特点。Nginx 的发展越来越成熟，其作为负载均衡服务器对外提供服务的方式既能在内部支持 Rails 和 PHP 程序，也能支持作为 HTTP 代理服务器。Nginx 的安装和配置简单。对网络稳定性的依赖小，能够承担高负载压力，可以通过端口检测到服务器内部的故障。使用 Nginx 可以适当地分配流量、流媒体等资源，动态地调整图像大小、缓存内容等等。Nginx 按照调度规则可以实现动态和静态页面的分离,Nginx 支持多种配置策略，可灵活的对后端服务器做负载均衡。目前，Nginx 的 upstream 支持 5 种负载均衡方式的分配：轮询、指定权重、IP 哈希、URL 哈希、fair。

(1) 轮询：轮询是直接默认的方式，每一个请求按顺序逐一分配到不同的后端服务器，如果后端服务器 down 掉了，则能自动剔除。

(2) IP 哈希：对同一个用户（即：相同 session 下）的请求，根据用户的 IP 地址计算的哈希摘要值与后端服务器的地址进行映射，这样该用户每次的请求就会由后端同一台服务器进行处理，有效避免集群下 session 不一致的问题。

(3) 权重：根据后端服务器的性能配置不同的负载权重，即后端服务器的性能高低与所配权重值成正比，使负载轮询的后端服务器得到更合理的利用，发挥出应有的性能。

(4) URL 哈希（第三方模块实现）：与 IP 哈希类似的原理，但此时参与运算的策略不再是 IP 地址的哈希摘要值,而是更长一串的完整请求 URL 地址，这种负载均衡策略对于需要做缓存的请求资源比较适用。

(5) fair（第三方模块实现）：将后端服务器的响应速度进行排序，响应耗时越小的服务器获得处理的优先级更高，这样表现出来的效果就是整个服务器集群的响应速度很快，处理业务更高效。

2.6 本章小结

本章围绕基于 web 前端性能优化模型，对相关技术做了详细介绍，包括 Json、Node.js 以及组件化开发思想、基于前端 MVC 的构建模型和主流前端框架技术，并介绍了自动化构建工具 webpack 和负载均衡组件。

下一章将对 web 前端性能优化关键技术进行详细介绍。

3 基于web前端的性能优化关键技术

本章主要阐述了基于web前端的性能优化模型和设计，分析总结了影响web前端性能的因素，然后详细描述了前后端分离框架模型设计总体思路，提出面向企业业务应用场景的组件化开发技术，并针对前端设计应用基于Ngixn的负载均衡算法，通过负载均衡算法的应用，将页面加载等压力分布到前端，减轻后台压力，提升开发效率，优化web应用性能。

3.1 基于web前端性能的影响因素分析

3.1.1 浏览器的工作原理

用户通过浏览器访问Web应用，因此研究web前端性能需要了解浏览器的工作原理。目前广泛使用的浏览器有IE、Chrome、Firefox。浏览器主要功能是通过向服务器请求资源并将用户选择的web资源渲染出来，资源的格式主要包括HTML文档、脚本、图片等。浏览器主要由以下几部分组成：

(1) 用户界面：即用户看到且可操作的界面，界面元素包括地址栏、前进/后退、书签栏等。
(2) 浏览器引擎：查询和操作渲染引擎的接口，用于控制浏览器和渲染引擎间的信息传递。
(3) 渲染引擎：将服务端返回的html内容、图片、样式文件等解析并显示出来。
(4) 网络：用来完成http等网络请求调用，具有跨平台的特点。
(5) 用户界面后端：用于绘制组合选择框及对话框等一些基本的浏览器窗口组件，根据不同浏览器所依赖的操作系统调用不同的界面绘制。
(6) JavaScript解析器：用于解析和执行JavaScript代码。
(7) 数据存储：为了不受浏览器关闭影响，浏览器通过数据缓存、Cookie、Local Storage等数据存储方法，实现在硬盘中保存各种数据。

如下3-1图是Chrome浏览器的结构图：

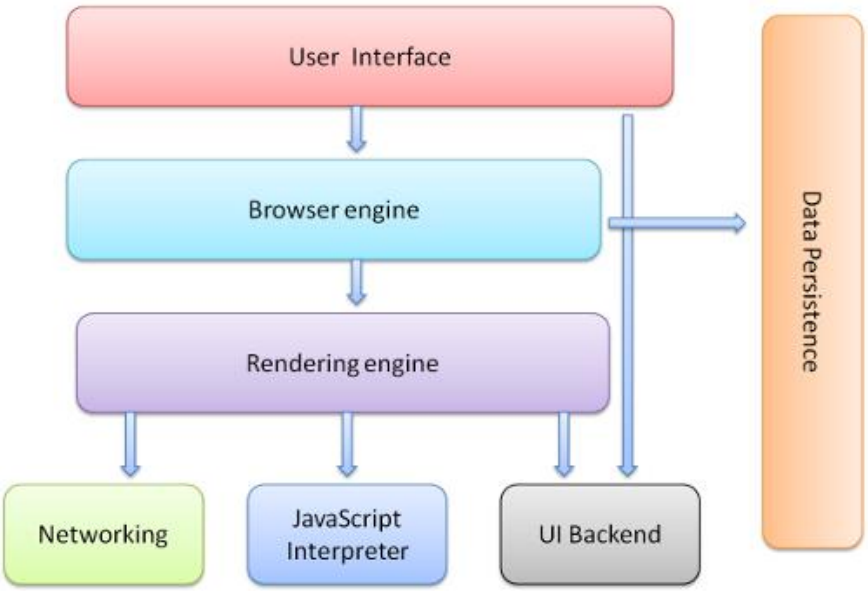

图 3-1 Chrome 浏览器的结构图

Fig.3-1 The structure of the Chrome browser

3.1.1.1 渲染引擎

渲染引擎（The Rendering engine）的作用是在浏览器窗口渲染所请求的内容，支持显示 CSS、HTML、图片等文件，渲染引擎的基本工作流程是首先通过网络获得所请求文档的内容，然后解析 HTML 并构建 DOM 树，然后构建 render 树，布局 render 树，最后是绘制 render 树。Chrome 浏览器使用 webkit 渲染引擎，webkit 渲染引擎在获取到文档内容后的渲染主流程如下图 3-2 所示：

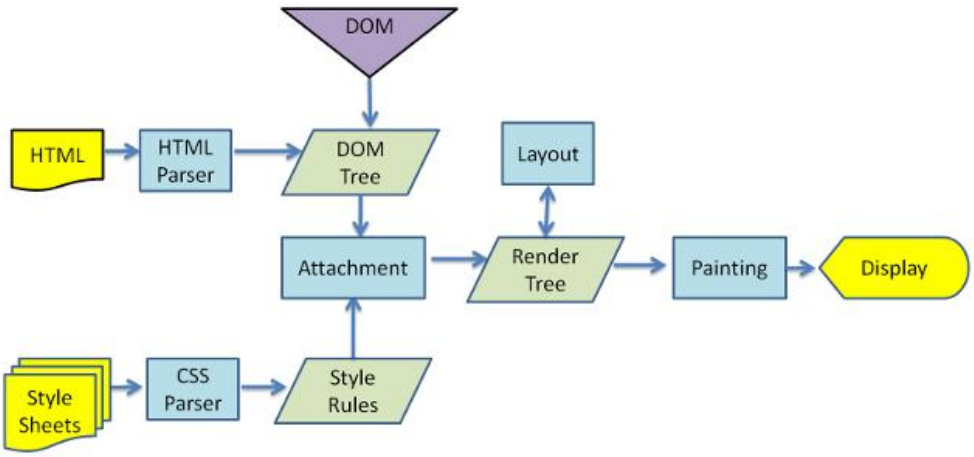

图 3-2 webkit 渲染引擎主流程图

Fig.3-2 WebKit rendering engine mainstream

渲染引擎开始解析 HTML，并将标签转化为内容树中的 DOM 节点，然后解析外部 CSS 文件及 Style 标签中的样式信息。这些样式信息以及 HTML 中的可见性指令将被用来构建另一棵 render 树（Render 树由一些包含有颜色和大小等属性的矩形组成，它们将被按照正确的顺序显示到屏幕上）。Render 树构建好之后，将会执行布局过程，它将确定每个节点在屏幕上的确切坐标。最后是绘制，即遍历 render 树，并使用用户界面后端绘制每个节点。为了更好的用户体验，渲染引擎将会尽可能早的将内容呈现到屏幕上，并不会等到所有的 html 都解析完成之后再去构建和布局 render 树，即解析完一部分内容就显示一部分内容，同时，可能还在通过网络下载其余内容。

3.1.1.2 解析与 DOM 树构建

解析是渲染中关键的一步，解析 HTML 文档即将其转换成程序可以理解的数据结构，解析结果通常以一个节点树表示，称为语法树或解析树。在 html 中，解析结束之后，浏览器将文档标记为可交互的，即用户可以操作界面。并且浏览器开始解析和执行处于延时状态的脚本文件，最后将 html 文档状态设置为完成，同时触发 onload 事件。浏览器对于 html 的解析是同步的，即浏览器在解析和执行一个 script 标签时会立即解析和执行该文件的内容，但是会停止执行 html 之后的内容，直到当前文件解析执行结束，这种方式，如果网页是通过外部方式引用脚本，会同步从网络获取资源，同时会阻塞文档

的解析直到资源被请求到。HTML5 增加了标记脚本为异步请求方式，当执行脚本时，另一个线程解析剩下的文档，并加载后面需要通过网络加载的资源，这种方式实现资源并行加载从而使整体速度更快。

3.1.1.3 渲染树构建

渲染树是由 HTML 文档中的可显示元素所组成的树结构，在完成 DOM 树的构建和样式表的解析后，渲染引擎根据信息生成渲染树，浏览器根据渲染树绘制内容。Webkit 将渲染树中的元素节点称为呈现对象，每个呈现对象都对应于屏幕上的一个矩形区域，每个渲染对象用一个和该节点的 CSS 盒模型相对应的矩形区域来表示，包含诸如宽、高和位置之类的几何信息。渲染树中禁止插入不可显示的元素，例如 head 元素。另外，display 属性为 none 的元素也不会在渲染树中出现。有一些 Dom 元素对应几个可见对象，它们一般是一些具有复杂结构的元素，无法用一个矩形来描述。例如，select 元素有三个渲染对象，分别为显示区域、下拉列表及按钮。一些渲染对象和所对应的 Dom 节点不在树上相同的位置，例如，浮动和绝对定位的元素在文本流之外，在两棵树上的位置不同，渲染树上标识出真实的结构，并用一个占位结构标识出它们原来的位置。

3.1.1.4 布局与绘制

布局是一个递归的过程，由根渲染对象开始，它对应 html 文档元素，布局继续递归的通过一些或所有的 frame 层级，为每个需要几何信息的渲染对象进行计算。当渲染对象被创建并添加到树中，它们并没有位置和大小，计算这些值的过程称为 layout 或 reflow。Html 使用基于流的布局模型，意味着大部分时间，可以以单一的途径进行几何计算。流中靠后的元素并不会影响前面元素的几何特性，所以布局可以在文档中从右向左、自上而下的进行。也存在一些例外，比如 html tables。

绘制阶段，遍历渲染树并调用渲染对象的 paint 方法将它们的内容显示在屏幕上，绘制使用 UI 基础组件。和布局一样，绘制也可以是全局的——绘制完整的树——或增量的。在增量的绘制过程中，一些渲染对象以不影响整棵树的方式改变，改变的渲染对象使其在屏幕上的矩形区域失效，这将导致操作系统将其看作 dirty 区域，并产生一个 paint 事件，操作系统很巧妙的

处理这个过程，并将多个区域合并为一个。Chrome 中，这个过程更复杂些，因为渲染对象在不同的进程中，而不是在主进程中。Chrome 在一定程度上模拟操作系统的行为，表现为监听事件并派发消息给渲染根，在树中查找到相关的渲染对象，重绘这个对象（往往还包括它的 children）。

3.1.2 影响 web 前端性能的因素

根据浏览器的工作原理，总结分析影响 web 前端性能的因素主要有 HTTP 请求数、响应时间、网页元素、web 缓存。因此，总结优化方向和优化手段[1][2]如下表 3-1 所示：

表 3-1 部分测试用例和测试结果

Table 3-1 Some test cases and test results

优化方向	优化手段
请求数量	合并脚本和样式表，CSS Sprites，拆分初始化负载，划分主域
请求带宽	开启 GZip，精简 JavaScript，移除重复脚本，图像优化，字体图标替代图片
缓存利用	使用 CDN，使用外部 JavaScript 和 CSS，添加 Expires 头，减少 DNS 查找，配置 ETag，使 AjaX 可缓存
页面结构	将样式表放在顶部，将脚本放在底部，尽早刷新文档的输出
代码校验	避免 CSS 表达式，避免重定向

通过表 3-1 可以看出，影响 web 前端性能的因素较多，过去的优化工作侧重“局部”，只解决了眼前的性能问题，而如何全面提取前端性能优化因素，并针对这些因素进行模型分析，并做出改进，尤其是提出适合大型 web 应用的优化策略是亟待解决的关键问题，也是本文要解决的重要问题。

3.2 前后端分离框架模型的总体架构

大型的 SaaS 产品其软件和相关的数据放在云端被集中式地托管，用户通常使用浏览器来访问服务，这样导致应用压力集中在后台运算，对 web 应用的性能要求较高。而目前，现有 web 系统主要采用划分元素或打开浏览器新窗口的方式来组织页面，虽然在浏览器中输入一个地址，但是子页面还是会整张页面的刷新，不同页面的公共组件也会被大量地重复加载，造成资源的

严重浪费，随着产品的功能复杂性的提升，嵌入网页的代码也越来越庞大，最终导致 web 应用的性能瓶颈，因此，为了适应现有 SaaS 产品的应用，SaaS 服务模式的开发必须从架构层面改进系统的性能。因此，本文前后端分离的总体架构模型如图 3-3 所示：

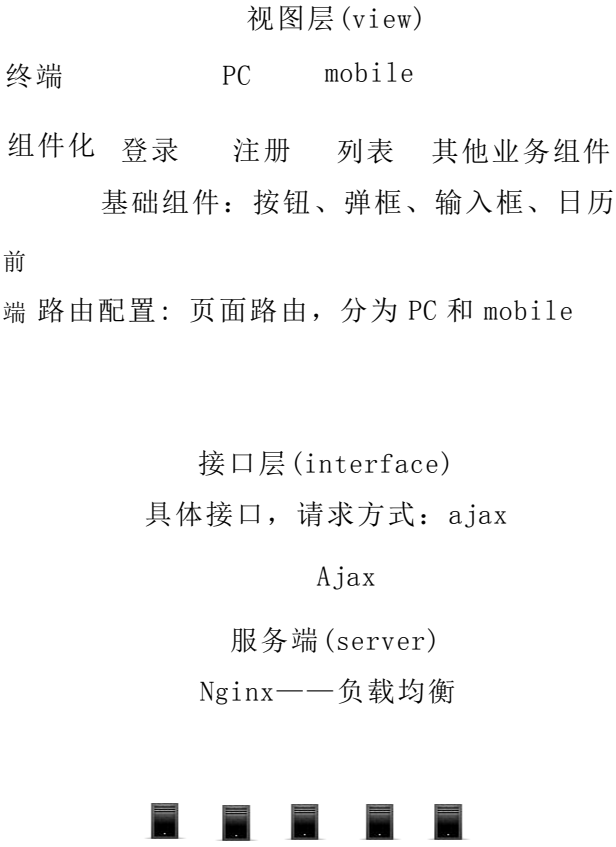

图 3-3 前后端分离的总体技术架构

Fig.3-3 Overall technical architecture of front and back separation

在前后端之间增加 nodeJs 作为中间层，后端通过 REST 框架实现前后端

分离架构。前端模型包括视图层、接口层和服务层，其中，视图层包含组件、路由模块；接口层包括所有具体的业务接口；前端的服务层，设置 Nginx 作为静态资源代理，并根据应用的性能要求，采用负载均衡的方案以达到高性能的要求，从而快速响应客户端的请求。后端服务主要通过 REST 框架，实现和前端服务层之间的接口数据传递。

该技术架构特点如下：

(1) 解耦前后端职责：Node.js 服务层统一归入前端工程中，相对系统之前的前端，本文叫“大前端”，分离之后，后端与前端只通过 restful api 接口进行连接，前端的路由配置等不在受限于后端，后端对外只需要关心提供的接口服务，将更多的精力放在后端服务的架构和性能上，脱离了前端的页面管理工作。

(2) 前后端开发由串行变为并行，提高整体开发效率：前后端分离之后，前后端通过接口进行直接联调，即使后端接口没有实现也不影响前端开发继续进行，前端可以通过调用 mock 数据先进行接口对接工作，同时后台进行接口的实际开发并进行单独的接口测试。前端完成之后，测试也可以按序进行。对于前端来说，后台接口开发完成之后，只是更换一下接口地址即可，从而实现前后端开发并行的效果。

(3) 实现多终端匹配，让各终端视觉效果和交互设计达到最优：前后端分离，前端基于 Node.js 可按照业务需求实现多终端匹配，并且通过面向企业业务应用场景的组件化开发模型，让各终端的体验效果和页面视觉效果达到最佳。

通过该技术架构，实现前端模块化和组件化开发模式，只需要通过 vue-router 管理用户请求和页面跳转；页面通过 ajax 调用后端的 restful api 接口；前端与后端定义一个相对通用的 JSON 响应结构，在开发完成以后，使用自动构建工具发布应用，帮助开发人员完成文件的合并压缩。这对于减少性能优化工作量是非常有帮助的。

3.3 面向企业业务应用场景的组件化开发技术

由于 web 应用中有大量重复使用的组件，因此通过研究面向业务应用场景的组件化开发，可以提升开发效率。而针对企业业务应用的 Web 前端的组件，不管这个组件是 UI 层级，还是 javascript 开发层级，都脱离不了该企业业务产品的模式，比如常见的登录、注册、文件操作等组件形式。因此本

小节通过对组件化开发技术进行研究，对组件进行分类处理，理清组件的应用边界，有针对性的积累和完善这些组件，最终形成一个针对某个业务组件的组件仓库，供开发者重复使用，而不需要重新开发，提高开发效率。

页面上的每个独立的可视/可交互区域视为一个组件,如下图 3-4 所示：

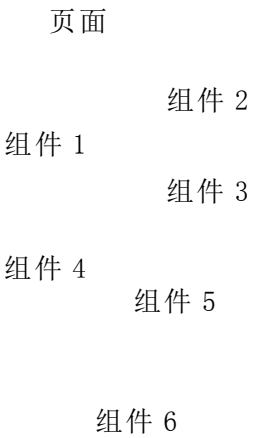

图 3-4 组件与页面的关系

Fig.3-4 The relationship between the component and the page

该结构利用组件高内聚、可重用、可互换、可组合的特点使得开发者之间不会产生开发时序的依赖，大幅提升并行的开发效率，也更容易支持多个团队共同维护一个大型应用的开发。

3.4 基于 Nginx 的负载均衡设计技术

前后端完全分离之后，系统划分成了前端和后端两大部分，暂且可把 NodeJs 划分为大前端部分，大前端通过 NodeJs 调用后端的 API，面向不同终端场景（比如：web 端、桌面客户端、手机 APP 端）向用户提供服务，由于单机 NodeJS 的性能瓶颈，无法满足来自各个终端和大批量的网络请求，所以需要在服务端进行横向水平扩展，即增加若干 NodeJS 前端服务器实例节点，多台 NodeJS 服务器分担相同的压力，从而解决前端服务器性能的压力，但对外只能有一个入口，所以必须添加一个负载均衡器 Nginx 来对外提供统一的入口，然后将前端接收到的各个终端的请求按负载均衡策略的算法分发到后端的 NodeJS 节点上进行相应的处理。

通过对 Nginx 负载均衡技术研究，本文结合 SaaS 应用架构，将 Ngnix

负载均衡设计为前端负载均衡及后台负载均衡。设计如图 3-5 所示：

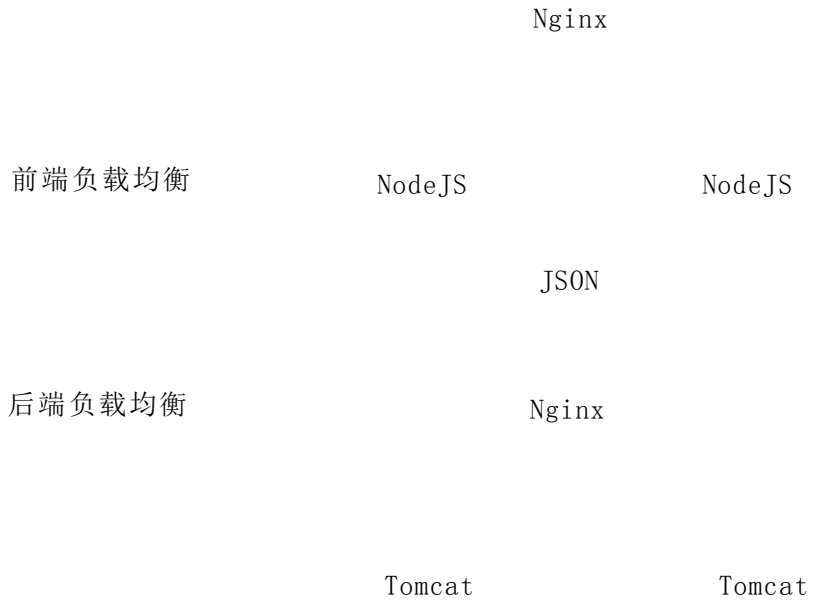

图 3-5 基于 Nginx 的负载均衡设计方案

Fig.3-5 Load balancing design scheme based on Nginx

该设计使前端服务器具备接收大量数据请求的能力，并且在某台 NodeJS 服务器宕机后，仍然能对外提供服务，因此，该类结构，本文定义为“前端负载均衡”的架构，该架构解决了前端性能问题。并考虑了真正业务处理的后台性能问题。该架构中，对于前端只负责页面渲染和数据的预校验作用，因此，前端服务器接收到的数据通过轻量级的数据交换格式 JSON，将请求数据传到内网的 Tomcat 业务服务器进行业务逻辑的处理。在前端业务的性能处理方面，来自前端大批量的数据传输到内网业务服务器，也会存在单机处理的性能问题，因此通过横向扩展来添加业务服务器，并在与前端交互的入口处添加负载均衡器 Nginx，原先页面的反馈路径从后端服务直接转移到 Node.js 服务端，缩短了访问响应时间，从而提升了系统对用户的响应速度。因此通过本文提出的“前端负载均衡”和“后台负载均衡”的这种设计结构，提升了服务器集群的抗压能力，增强了集群的容错能力，最终，保证提供给用户快速，高效，稳定的互联网服务体验。

3.5 本章小结

本章主要通过分析web前端性能的关键因素，提出了前后端分离框架模型设计的总体思路，并将该框架特点进行了总结。接着，面向企业业务应用场景提出组件化开发设计思路，提升前端开发效率。最后，针对前端设计提出应用Nginx的负载均衡的前端设计方案，实现了解决多终端性能、组件化开发和打包部署的完整的开发模型，减少前后端耦合性，进一步提升web应用性能。

下一章内容主要是将本章模型架构及技术用于SaaS业务系统中进行实践，通过实践反馈，证明该模型的实践可行性和开发的高效性。

4 面向SaaS应用的性能优化框架模型的实践

本章将web前端性能优化模型应用到了一款SaaS级的企业web应用，原web应用系统采用普遍使用的模板引擎进行页面渲染，后端控制网页路由，在这种开发模式下，前后端工作无法独立，再次迭代和维护效率较低，且对后端服务器造成一些不必要的压力，同时网页性能优化受到限制。本章详细描述了前后端分离框架模型和组件化开发方案如何应用到该系统中，并对负载均衡方案的应用做了详细的说明，通过测试证明了该模型的可行性和高效性，并对前后端开发效率和网页性能的提升起到重要的影响。

4.1 SaaS应用的业务介绍和概要设计

该SaaS应用是一款电子签名产品，基于合法电子签名，在云端完成电子合同的缔约，签署流程自动化流转。不仅合法合规，而且高效便捷，助力企业实现全程数字化办公。其面向多终端用户提供服务，Web端只是该系统中的一部分，Web端与其他终端（比如APP端）形成整体的用户端，后端支撑用户端的所有服务。

4.1.1 需求分析

根据系统业务介绍，可以分析得出本系统的Web端不仅要提供面向电脑屏幕的页面，而且需要提供面向手机屏幕的网页，十分注重用户体验。Web端功能模块包括登录、注册、文件、文件签署、文档预览、签章管理、个人信息设置、实名认证等核心功能，信息架构如下图4-1和图4-2所示。

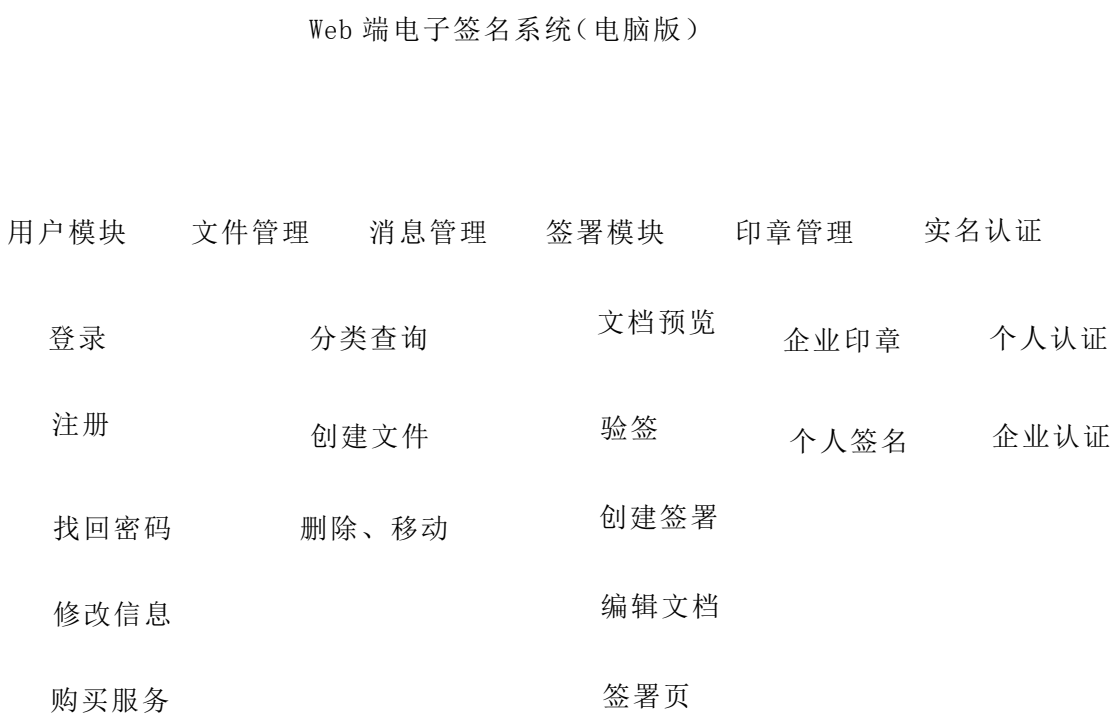

图 4-1 web 电脑版系统信息架构

Fig.4-1 Web computer system information architecture

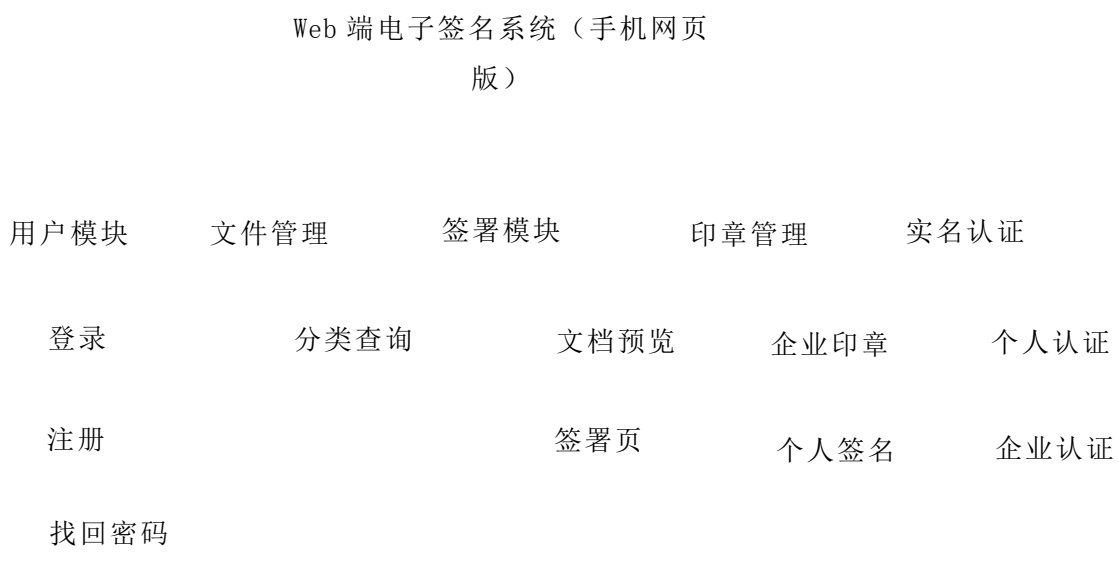

图 4-2 web 手机网页版系统信息架构

Fig.4-2 Web mobile web page system information architecture

系统核心流程，新用户注册系统，登录系统之后完成个人实名认证，实名认证通过之后可以发起签署流程，创建签署，上传需要签署的文件，设置签署流程等信息，然后确定发出文件，系统将签署消息自动推送到签署者的邮箱或手机短信中，文件按照顺序自动流转到各签署方，签署方可通过邮件或短信中的链接直接访问到签署文件，或者直接登录系统处理需要“待我签”的文件，其中短信链接进入到 H5 页面，终端场景是手机端，所有签署方完成签署之后，本次签署即完成，流程中的相关人可以登录系统对文件进行查看、下载等操作，核心业务流程图见图 4-3 和图 4-4：

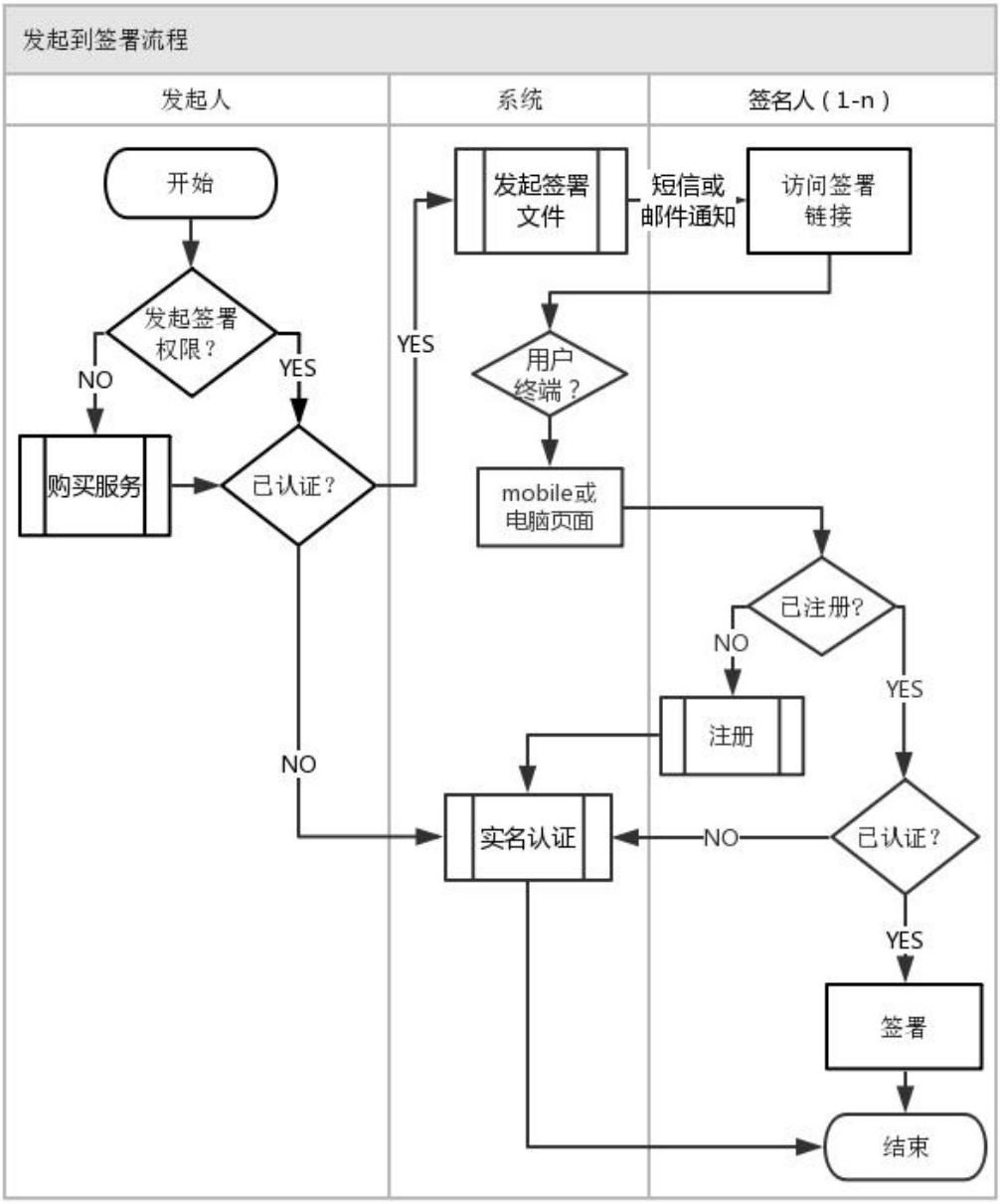

图 4-3 发起到签署结束的业务流程

Fig.4-3 Start to sign the end of the business process

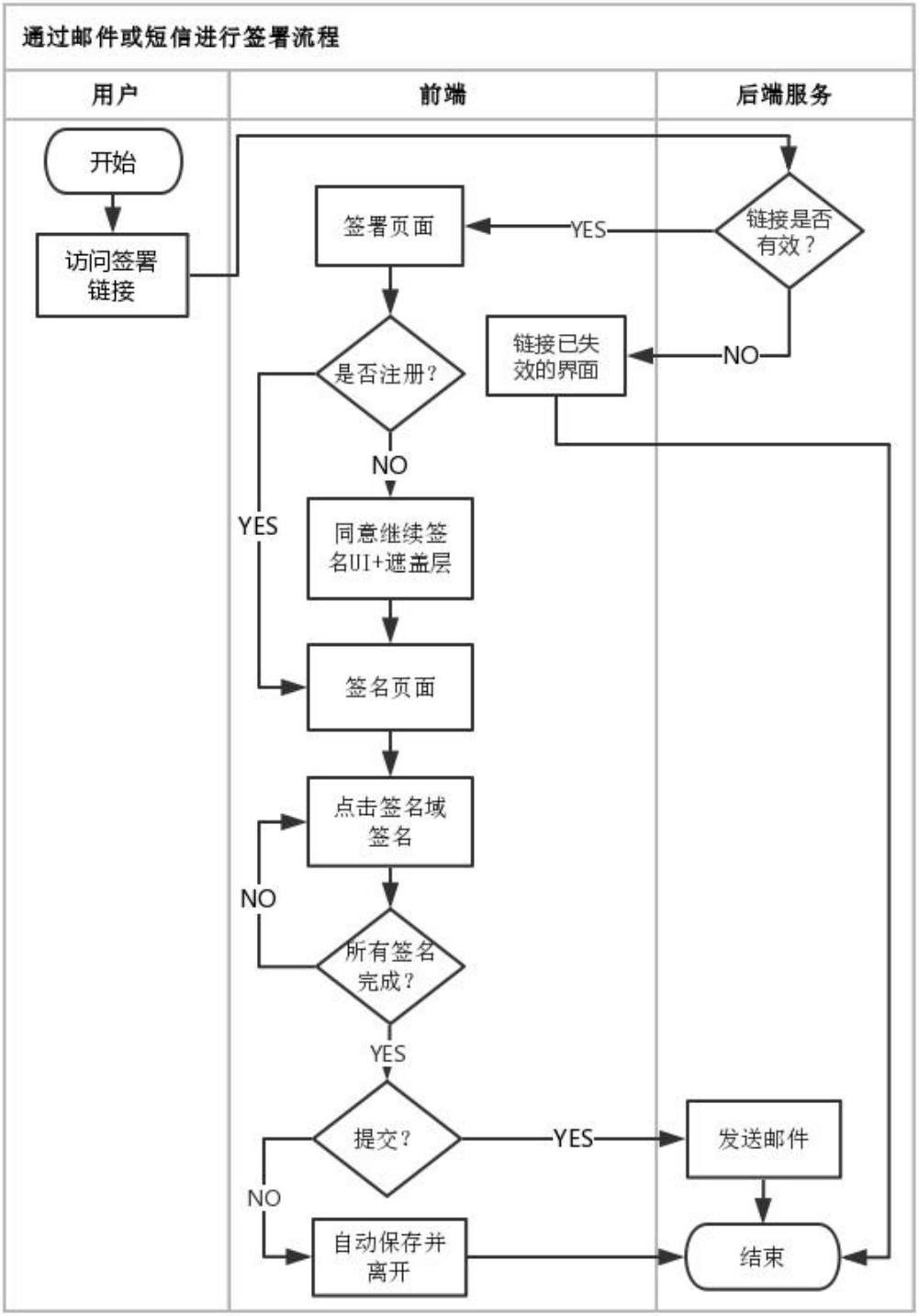

图 4-4 签署人通过邮件或短信进行签署的核心流程

Fig.4-4 The core process of signatories signed by mail or SMS.

由此可见，该应用的使用场景的复杂性，直接给 web 端页面对终端的适配提出高要求，不仅要保证页面性能良好，而且要使页面的视觉美观、功能完整且可用性好。

4.1.2 概要设计

电子签名系统的功能模块主要分为 5 大模块：用户模块、文件模块、签署模块、印章模块、实名认证模块。

(1) 用户模块：用户模块主要包含对用户信息的增、删、查、改，包括用户个人信息的查询，用户数据修改，实现包括登录、注册、找回密码、修改密码、修改个人信息等功能。数据库相关设计表如表 4-1 所示。

(2) 文件模块：文件模块主要是对签署文件信息的查询和处理，包括查询不同类型的文件、单个签署文件详情查询、文档数据查询、对文件检索分类等操作。同时对数据访问权限做了限制，要求数据的访问和操作权限只有系统内有权限。数据库相关设计表如表 4-2 所示。

(3) 签署模块：签署模块主要包含对签署文件的处理，包括签署文件中信封接收者信息的查询、修改，对签署文件表单项的增、删、查、改等操作。同时对数据访问权限做了限制，要求数据的访问和操作权限只有系统内有权限。数据库设计表信封接收者信息如表 4-3 所示，表单项数据设计如表 4-4 所示。

(4) 印章管理模块：印章管理模块主要包括印章的创建、查询、修改、删除、访问权限等控制。同时对数据访问权限做了限制，要求数据的访问和操作权限只有系统内有权限。数据库相关设计表如表 4-5 所示。

(5) 实名认证模块：实名认证模块主要包含实名认证信息、个人和企业信息、认证结果等查询、认证信息修改等操作。同时对数据访问权限做了限制，要求数据的访问和操作权限只有系统内有权限。数据库相关设计表如表 4-6 所示。

表 4-1 用户信息数据库表 User

Table 4-1 User information database table

权限	(属性)类型	名称	备注
private	bigint(20)	id	系统用户的 id
private	varchar(50)	username	用户的昵称

private	varchar(100)	email	用户的邮箱，可用于登录
private	varchar(50)	phone	用户绑定的手机号码,可用于登录
private	varchar(250)	password	登录密码
private	int(11)	age	用户的年龄
private	varchar(50)	location_code	用户所在的位置
private	varchar(10)	sex	用户的性别
private	datetime(0)	regist_date	用户的注册日期
private	int(10)	regist_from	用户注册来源（1-Web 网页，2-PC 客户端，3-移动 Android 终端，4-移动 IOS 终端，5-第三方登录）
private	int(10)	level	用户级别（0-未认证用户（basic），1-认证用户（standard），2-认证可信用户（business），3-....）
private	varchar(100)	real_name	用户的真实姓名
private	datetime(0)	last_login_date	最近一次登录日期时间
private	varchar(16)	ip	用户 IP
private	varchar(1000)	portrait	用户头像路径
Private	tinyint(1)	activated	账号激活状态（0-未激活，1-已激活,2-重设账号后待激活）

表 4-2 信封列表查询 EvInfoQueryResp

Table 4-2 Envelope list query

权限	(属性)类型	名称	备注
private	String	envelopewsid;	信封全局 id
private	String	sender_wsid;	信封发件人用户的全局 id
private	String	sender_name;	信封发送者名称
private	Integer	type;	信封类型
private	String	title;	信封标题

private	String	subject;	信封主题
private	Integer	secure_level;	信封安全级别
private	Integer	status;	信封状态
private	Date	created_datetime;	信封创建的时间
private	Date	timer_datetime;	定时启动的时间
private	Date	send_datetime;	信封发送的时间
private	Date	pire_datetime;	信封过期的时间
private	Date	status_datetime;	信封所处状态的时间
private	Date	finished_datetime;	信封完成的时间
private	String	status_reason;	信封所处状态的原因
Private	String	metadata;	信封信息元数据

表 4-3 信封中接收者信息表对应的映射对象 EvRecipientInfo

Table 4-3 The mapping object EvRecipientInfo corresponding to the recipient information table in the envelope

权限	(属性)类型	名称	备注
private	Long	id;	数据库自增 id
private	String	wsid;	接收者全局 id
private	String	envelope_wsid;	信封全局 id
private	String	name;	接收者名字
private	Integer	type;	接收者类型
private	Integer	status;	当前接收者的处理状态
private	String	assigned_message;	分配给当前接收者的消息
private	Integer	assigned_sequence;	分配给接收者初始的信封处理顺序（固定不变）：1~n，即：处理优先级(数字相同则表示同时处理)
private	Integer	sequence;	接收者当前处理信封的实时顺序（动态变化）：1~n ，即：处理优先级(数字相同则表示同时处理)
private	Date	view_datetime;	浏览信封时间

private	Date	handle_datetime;	接收者处理信封的时间
private	String	handle_reason;	接收者处理信封的原因
private	String	contact_metadata;	接收者的联系方式元数据
private	String	author_wsid;	授权者 recipient 的全局 id

表 4-4 表单查询响应信息 EvFormInfoQueryResp

Table 4-4 Form query response information

权限	(属性)类型	名称	备注
private	String	form_wsid	表单全局 id
private	String	envelope_wsid	表单所属的信封全局 id
private	String	recipient_wsid	表单所属接收者全局 id
private	String	file_wsid	表单对应的文件全局 id
private	String	form_metadata	表单元数据信息

表 4-5 印章管理设计表 Seal

Table 4-5 Seal management design table

权限	(属性)类型	名称	备注
private	bigint(20)	id	印章的 id
private	varchar(100)	name	印章的名称
private	double	size	印章大小（单位：KB）
private	int(10)	seal_from	印章来源（0-默认生成方式，1-上传图片方式，2-手写方式，3-输入文字方式）
private	longblob	data	印章信息
private	varchar(100)	file_id	印章在文件数据库中 id
private	int(10)	remain_state	印章保留状态（0-一直保存；1-签名后删除）
private	int(10)	check_state	审核状态（0-已审核、1-未审核、2-未通过审核）
private	datetime	time	印章产生日期
private	bigint(20)	user_id	所属者 id 号

private	datetime	check_date	审核日期
private	varchar(200)	serial_code	印章序列号
private	varchar(255)	md5	签章文件 md5

表 4-6 个人实名认证设计表 PersonProve

Table 4-6 Personal real name certification design table

权限	(属性)类型	名称	备注
private	bigint(20)	id	自增序列号
private	bigint(20)	user_id	用户的 id 号
private	varchar(100)	name	用户姓名
private	varchar(50)	phone	手机号
private	varchar(50)	email	邮箱
private	int(11)	id_card_type	身份证类型（0：二代身份证；1：临时身份证）
private	varchar(50)	id_card_code	身份证号
private	int(10)	id_card_valid	用户身份证有效时长(0-长期；1-十年；2-二十年)
private	datetime	id_card_valid_date	身份证有效截止日期
private	varchar(1000)	id_card_pic1	用户身份证正面照
private	varchar(1000)	id_card_pic2	用户身份证反面照
private	datetime	submit_date	提交验证时间
private	datetime	pass_date	验证通过时间

4.2 前后端分离模型实践

电子签名系统采用第三章提出的前后端分离框架模型，将系统整体架构分为“大前端”和后端，首先在电子签名系统的前端和后端之间引入 nodeJs 作为中间层，后端通过 REST 框架实现前后端分离架构，在工程开发过程，实现前后端职责解耦，解决了前后端协作问题。

4.2.1 电子签名系统的前端架构的设计

电子签名系统的前端整体架构设计分为视图层、接口层和服务层，其中视图层包含 UI 组件和页面路由模块；接口层包括所有具体的业务接口；服务层包含 nginx 作为静态资源代理。前端开发选用轻量级的 vueJs 作为基础库，采用 vue-router 对页面路由进行构建和管理，实现 vuex 管理组件状态信息，同时，前端基于 nodeJs+npm+webpack 对开源依赖包和工具实现前端模块化打包、合并和压缩文件。前端主要开发技术如下图 4-5 所示：

前端开发：vueJs+vue-router+vuex

自动化管理工具：nodeJs+npm+webPack

图 4-5 前端主要开发技术

Fig.4-5 Main development technology of front end

(1) 前端视图层：视图层由 UI 组件和页面路由模块构成，其中 UI 组件包含基础组件库和业务组件，页面路由，主要实现了页面路由与 UI 组件的逻辑联系，并且实现多终端的匹配。前端视图层具体设计如下图 4-6 所示：

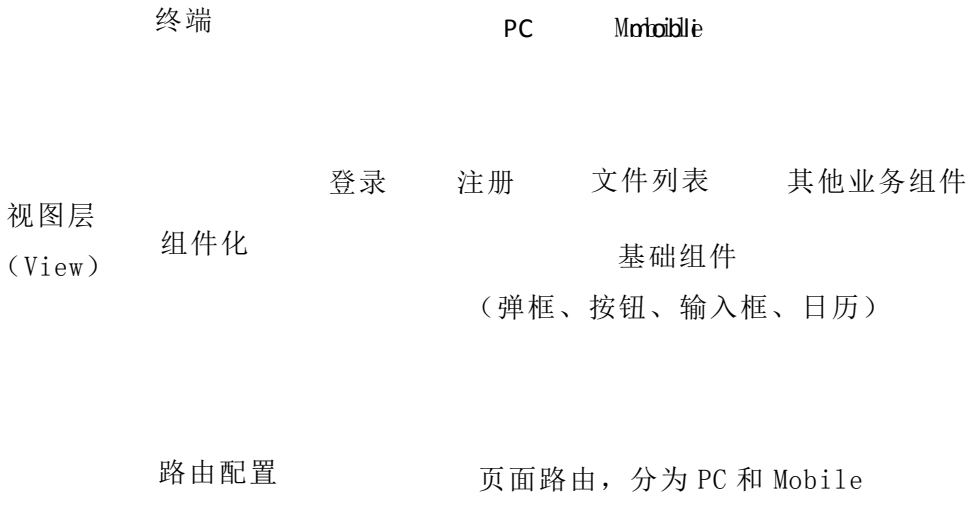

图 4-6 前端视图层设计图

Fig.4-6 Front view layer design map

本应用要求 Web 应用要适配各种终端，包括电脑或移动终端，因此，在页面路由的设计中引入面向多终端的设计方案，即：用户通过浏览器发起 URL 请求，通过路由将 URL 进行路由匹配，URL 一致且匹配成功，再通过 UA（User Agent）检测请求来自的终端类型进行终端页面的匹配，请求进入 Node.js 服务器上进行处理之后将对应的页面渲染输出到相应的不同终端的浏览器。其具体流程图如图 4-7 所示：

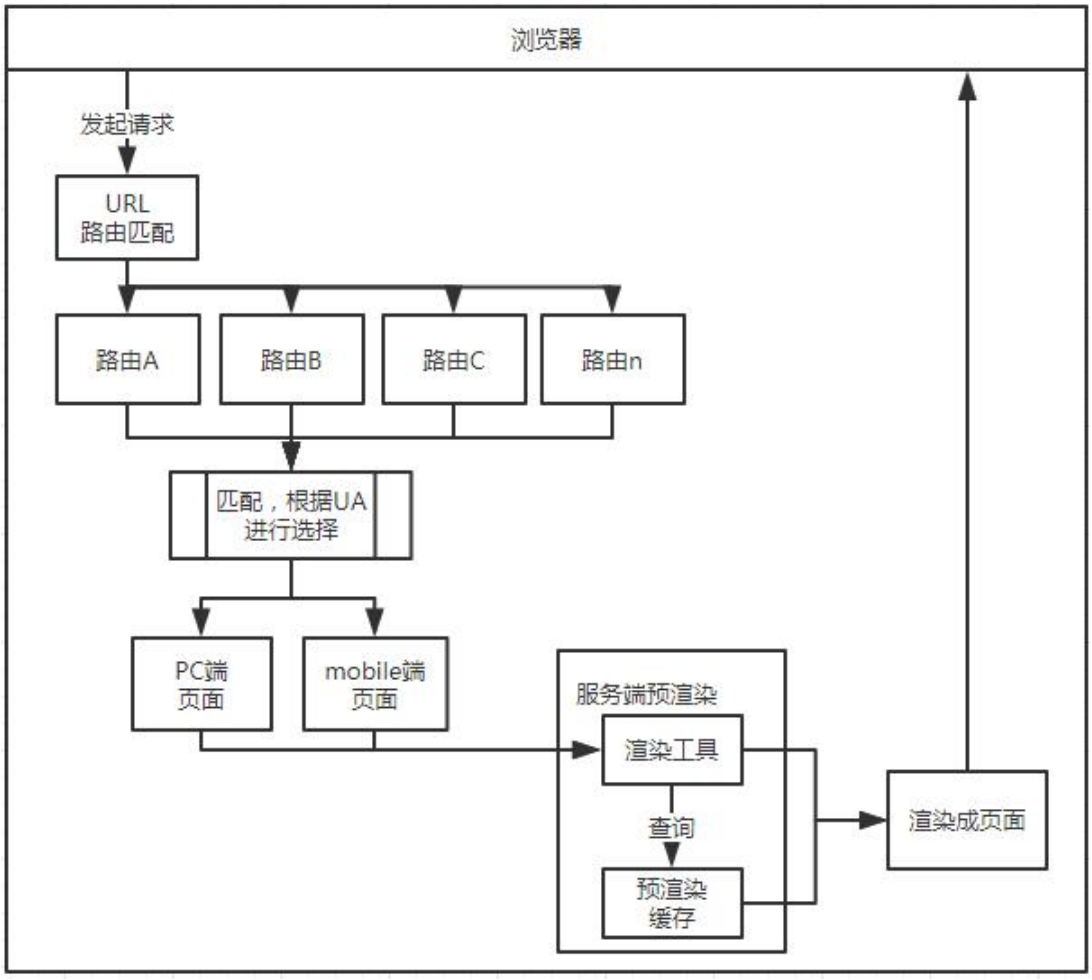

图 4-7　页面请求处理流程图

Fig.4-7　Page request processing flow chart

(2) 前端接口层：前端接口层是前端视图层和前端服务层的中转站，本文也将其看做 javascript 层面的组件，其接口可提供任意终端调用，向界面输出统一的数据结构，封装请求方法对应后台接口类型分为 POST、GET、PUT、DEL 四种类型，接口层通过 Ajax 与 Node.js 服务端进行连接，将请

求传到服务端，服务端处理之后将结果通过 Ajax 返回给接口层，接口层将处理结果最终返回给视图层。接口层的具体设计如下图 4-8 所示：

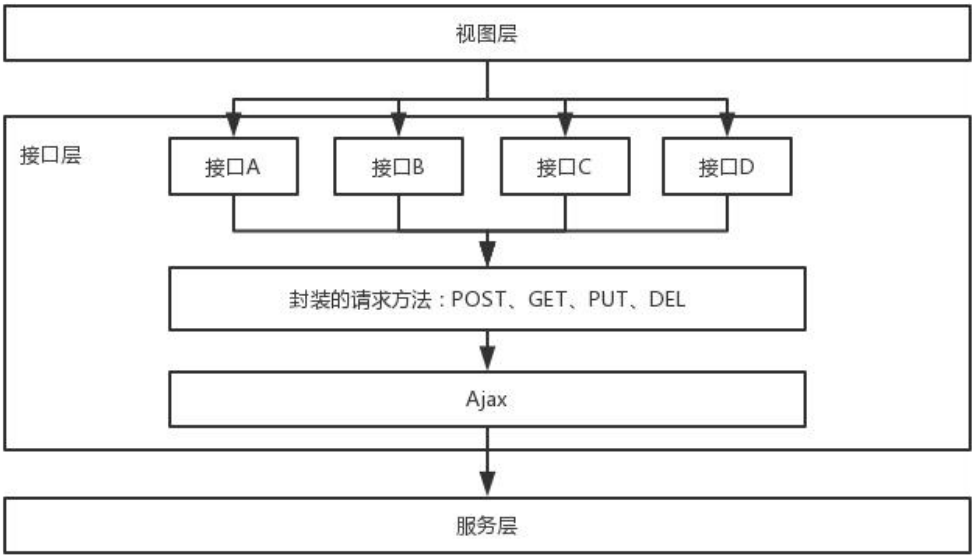

图 4-8 接口层具体设计实现

Fig.4-8 Concrete design and implementation of interface layer

(3) 前端服务层：该层直接与前端的接口层连接，处理来自用户端的请求并将处理结果返回给接口层或浏览器端，同时通过该层调用后端提供的接口，请求获取业务数据。并且在服务层可将 Nginx 作为静态资源代理，服务层可以根据应用的性能要求按需扩展，采用基于 Nginx 的负载均衡方案以达到高性能的要求，从而快速响应客户端的请求。前端服务层的 Nginx 的负载均衡设计图如下图 4-9 所示：

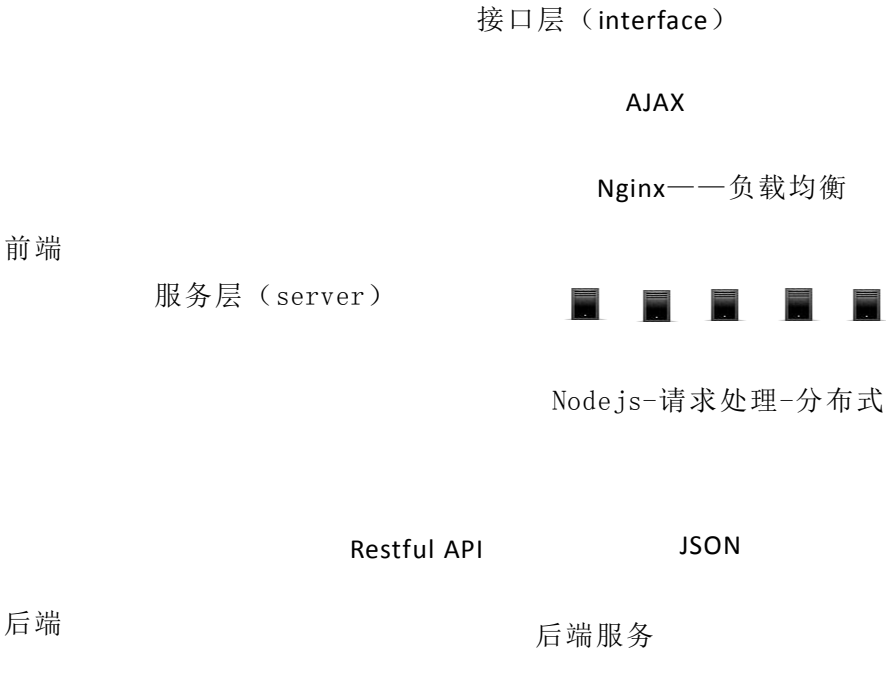

图 4-9 前端的服务层

Fig.4-9 Front-end service layer

4.2.2 电子签名系统的后端架构的设计

本应用的后端采用经典的 MVC 模型设计，相比于前后端分离之前，现在的后端剥离了前端页面，其对外只提供 API，因此，以 MVC 的设计模式，视图层不是页面渲染，而是 API 接口，后端的整体设计上分为 API 接口层、控制层和数据层。

(1) API 接口层：后端的 API 接口层将后台处理后的数据对象序列化成 JSON 格式的文本数据，向上和对外提供接口服务，通过 HTTP 协议传输给前端进行数据的解析并渲染。

(2) 控制层：控制层主要用于电子签名系统的业务逻辑处理，所有业务逻辑都在控制层实现。控制层向下数据层请求业务数据实现业务功能，将处理后的数据向上传给 API 接口层，从而完成业务逻辑的处理。

(3) 数据层：数据访问层通过数据持久层访问数据库数据，持久层，使用 ORM 框架（如：mybatis），将数据字段映射成相应的后台开发语言（如：java）中的对象，这样对数据库的“增、删、改、查”的操作就变成了对相关

对象的操作，控制层提交的待处理业务数据就能在持久层中顺利的进行处理，持久层通过编程式的代码逻辑语言将业务逻辑数据通过ORM框架持久化放到数据库中，完成最终的业务逻辑的实现。

4.3 组件化开发方案的应用

系统采用第三章提出的组件化开发技术，对前端的开发进行分层设计，使得不同程度的开发人员分配到对应难度的开发任务，并且可满足多终端设计中，让各终端的视觉效果达到最优。实践开发中，采用Vue.js作为前端基础库，通过双向数据绑定连接View（视图）和Model（模型）层。实际的DOM（Document Object Model，文档对象模型，简称DOM）操作被封装成Directives（指令）和Filters（过滤器）。通过采用组件化开发技术，具有轻量级、高性能、灵活性的特点，适合移动场景，正好符合本SaaS应用面向多终端设计的要求。

4.3.1 电子签名系统业务组件设计

基于面向企业业务场景的组件化开发模型，应用在电子签名系统中，为了让各终端的体验效果和页面视觉效果达到最佳，将视图层分为了移动端和电脑客户端。UI组件化的设计思路，首先将电子签名系统的所有页面通过原型的方式图形化表示，如下图4-10所示：

图 4-10 界面原型

Fig.4-10 Interface prototype

然后根据界面原型进行 UI 组件划分，通过对所有界面进行分析、解构、重组，最后总结 UI 组件的构建分成移动 UI 业务组件、公共 UI 业务组件、电脑 UI 业务组件 3 大部分。经过详细的需求分析之后，分别对移动端网页和电脑端网页做了原型设计，根据所有界面原型进行组件分析、划分和抽象，最终实现电子签名系统 UI 组件的划分和抽象，UI 组件的总体构建思路如下图 4-11 所示：

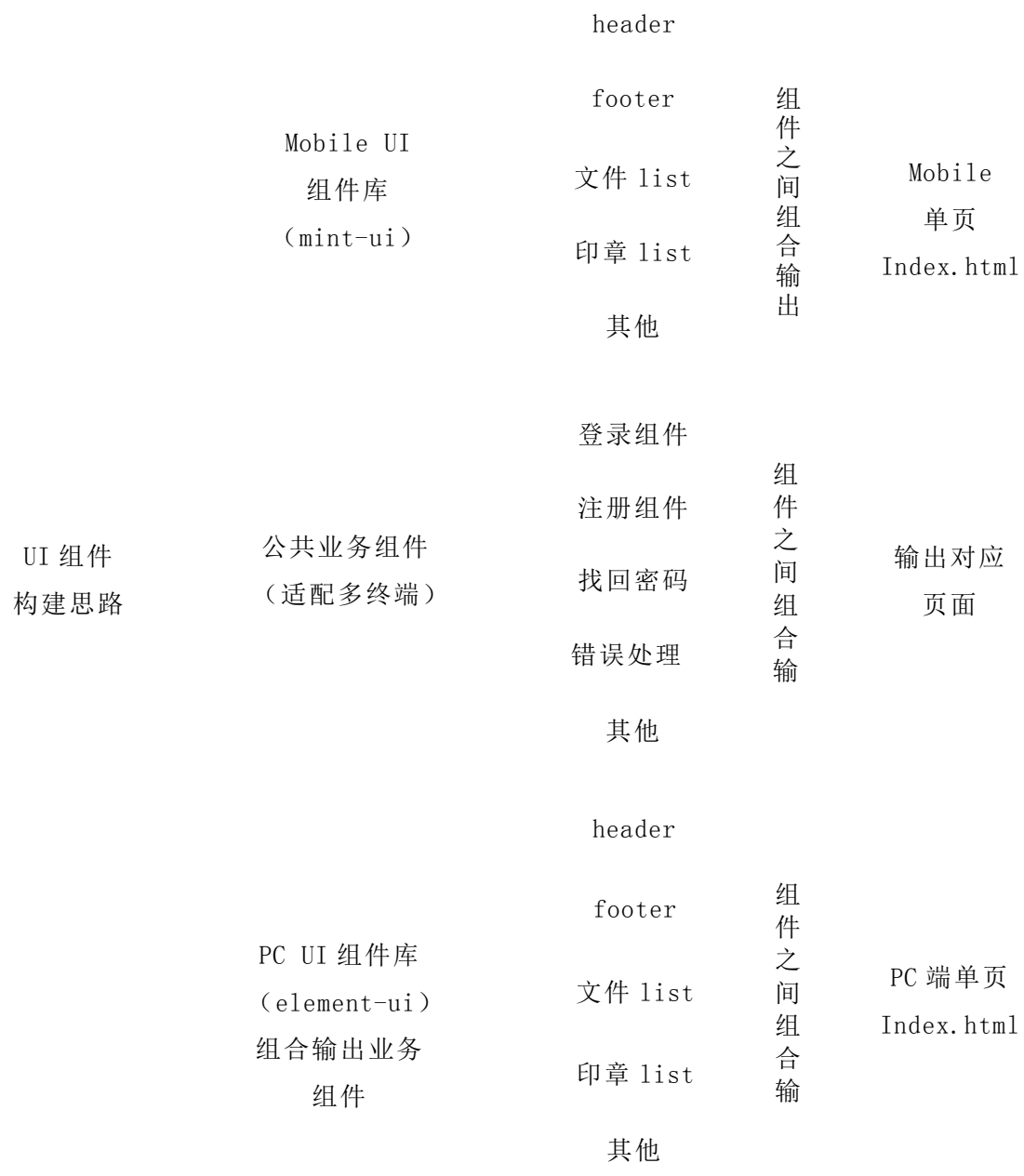

图 4-11 UI 组件的总体构建思路

Fig.4-11 The general idea of building UI components

为了简化开发量，本文分别选择了 mint-ui 和 element-ui 作为移动端和电脑端业务组件的底层 UI 组件库，底层 UI 组件库导入开发工程之后，只需要升级维护。业务组件库由前端开发人员根据实际业务需求进行实现，移动端 UI 业务组件库和电脑端 UI 业务组件在工程上分别存放于不同的文件，对

于工程的后期维护和功能的扩展十分方便，也便于管理和任务的分配。

(1) 移动 UI 业务组件：移动 UI 业务组件面向移动网页场景提供业务组件，组件包括移动端页面的公共头部、底部、文件列表、印章列表、裁剪等业务功能组件，各组件独立存在，同时可以组合成一个新的页面或功能模块，最终被 web 服务端渲染到浏览器。本系统中选用 mint-ui 作为业务组件的底层 UI 组件，便于开发快速实现业务组件的封装。构建思路如下图 4-12 所示：

图 4-12　移动端 UI 业务组件构建思路

Fig.4-12 Construction idea of mobile terminal UI service component

(2) 公共 UI 业务组件：公共 UI 业务组件面向不同终端可自动适配的业务组件，通过抽象出来形成可在任意终端进行适配的组件，合并提供相同功能和视觉效果的组件，同时减少开发量，这类组件开发人员只需要提供和维护一套。公共 UI 业务组件包括登录、注册、找回密码、错误（诸如：404、403、500 错误）等业务功能组件，这类公共组件的一个特点是供系统外部页面使用，各组件可以独立存在，同时也可以组合成一个新的页面或功能模块。此部分的 UI 实现由开发人员根据实际业务需求完成，不依赖于任何底层 UI 组件，页面多终端适配的条件起到一定的约束作用，所以 UI 由开发人员实现对于以后的维护和扩展都是最合适的选择。构建思路如下图 4-13 所示：

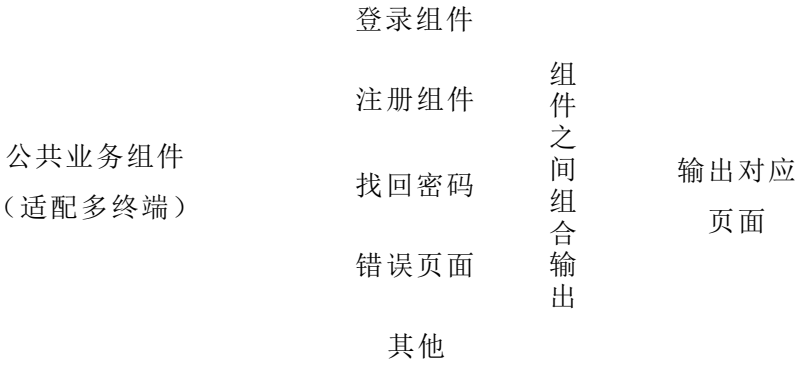

图 4-13 公共 UI 业务组件构建思路

Fig.4-13 The idea of building public UI service components

(3) 电脑 UI 业务组件：电脑 UI 业务组件面向电脑端网页场景提供业务组件，UI 业务组件包括页面的公共头部、底部、文件列表、印章列表、裁剪等业务功能组件，各组件可以独立存在，同时也可以组合成一个新的页面或功能模块，最终被 web 服务端渲染到浏览器。本系统中选用 element-ui 作为电脑 UI 业务组件的底层 UI 组件库，便于开发快速实现业务组件的封装。构建思路如下图 4-14 所示：

图 4-14 电脑端 UI 业务组件构建思路

Fig.4-14 The idea of building UI service components at the computer end

4.3.2 电子签名系统组件化开发实现

结合前后端分离模型，组件化开发思想，进而将应用的前端工程目录结

构按照视图层、接口层和服务层进行划分，随着工程的逐渐庞大，其带来的好处将不仅是表现在让工程文件分类更清晰，而且将减轻前端对工程维护的压力。根据上一节对电子签名系统UI组件的划分，本应用的前端工程目录6大文件组成，分别为commons、components、interfaces、page-components、routes、styles，目录的划分最终的目的是便于管理和维护，本系统的前端工程整体划分结构如图4-13所示。

(1) Commons文件：Commons文件用于存放公共文件，包括示例文件、图片、第三方资源等文件，可提供其他诸如css、javascript、html文件调用。同时也方便开发人员对这些文件的维护。

(2) Components文件：Components文件用于存放UI业务组件，包括移动UI业务组件、公共UI业务组件、电脑UI业务组件,系统所有独立的业务组件统一进行管理，方便维护。

(3) Interfaces文件：Interfaces文件用于存放前端封装的数据处理接口，映射到模型中的接口层，面向组件化的开发思想，实质可以看做是javascript层面的组件，其作为视图层和服务层的桥梁，向下获取从服务层返回的结果，进行处理之后，先上向视图层输出统一的数据结构，其提供的接口可提供各终端进行调用。

(4) page-components文件：page-components文件用于存放页面，页面内部由多个基础业务组件组合成一个完整的功能页，最终组成客户端可访问的页面，即浏览器端请求访问的页面，比如错误页面、登录页面、注册页面、找回密码页面、文件页面、个人信息页面等。

(5) Routes文件：Routes文件用于存放路由配置文件，实现路由的分发，为了让不同终端的用户体验效果达到最佳，这里的重构设计将移动端单独做了一份页面，但是基于组件化的思想，各终端共用了基础的业务组件，共用接口层，只有view层做了变化，因此，达到在保证各终端体验效果最佳的情况下，最大化地减少功能重复开发的成本。

(6) Styles文件：Styles文件用于存放样式模块，包括css、less、fonts等文件。方便开发人员后期的统一维护。

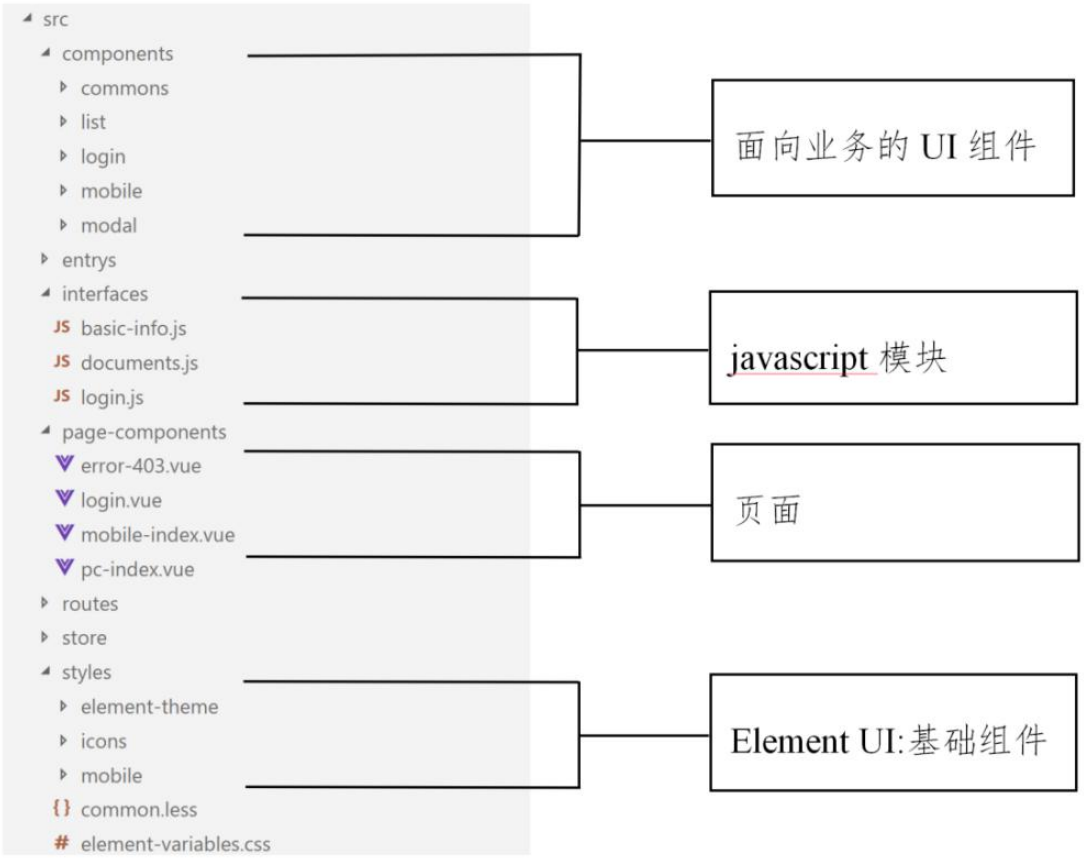

图 4-15 前端工程整体划分图

Fig.4-15 Whole division chart of front end Engineering

根据原型划分的业务组件，通过 hmtl、css、vue 等对组件进行编码实现，业务 UI 组件实现以电脑端网页版的左边导航栏组件为例，该组件在工程中作为一个独立的业务组件存在于 Components 文件里面，按照业务需求，左边导航栏组件中的导航菜单包含三个，分别为文件中心、印章管理和帐号设置，该导航组件菜单栏以后可以按需进行扩展，添加菜单项即可，作为单独的文件，可以独立维护，也能同其他组件组合成新的组件，不影响其他组件和页面的使用，基于 vue 实现，其核心代码如下所示：

```
<!-- 左边列表组件 -->
<template>
  ....
            <li v-for="value,key,index in listItems" class="list-group" >
```

```
                    <router-link :to="value" :class="{'list-group-item-active':index==
=listActiv                          e}" class="list-group-item">
                        <i class="icon" :class='iconList[index]'></i>
                        <span class="list-group-item-cont">{{key}}</span>
                    </router-link>
                </li>
            .....
    </template>
    <script>
        ...
                    listItems:{
                        '帐号设置':'/user/info',
                        '印章管理':'/user/info/seals/seal-admin',
                        '文件中心':'/user/info/files'
                    },
                    iconList:[
                        'icon-user','icon-signature','icon-document'
                    ]
        ...
    </script>
```

按照此开发思路，对业务组件中的文件列表组件、头部导航、底部导航等进行实现，然后通过组件之间的组合，形成文件页面、登录页面、注册页面、签署页面、印章页面等，组件实现原型图如图 4-16 所示。使用组件化开发思想最终实现电子签名系统电脑端的页面和移动端的页面。

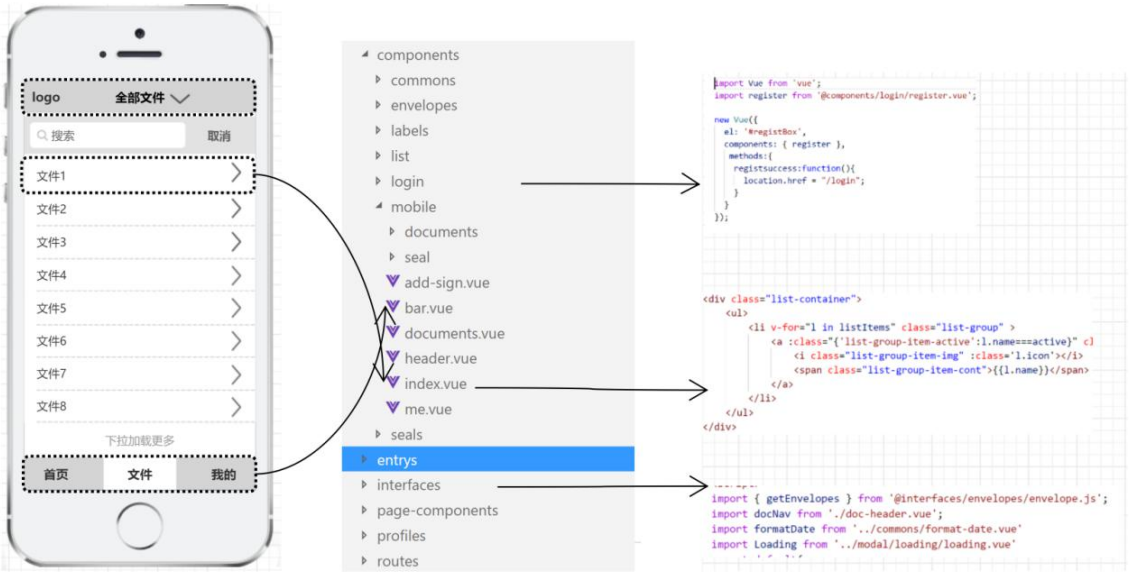

图 4-16 移动端文件页面组件与实现的构建图

Fig.4-16 Building diagram of mobile end file page components and Implementation

用户在手机端收到电子签名系统推送的短信通知，通过访问短信中的签署链接，在手机浏览器中访问 H5 页面，登录系统选择待签署的文件，选择签署，进入签署页面，进行签署等操作，签署完成之后，可以继续浏览已签署的文档或者在文件列表页面，查看其他文件详情等操作，移动端的文件列表页面和签署页面根据业务需求，视觉效果和交互设计根据移动端场景量身设计，让用户在移动端网页中的体验效果做到最佳。同时，电脑移动端的用户可以通过电子签名系统推送的邮件消息访问系统，登陆系统之后，进入签署页面进行签署等操作，提交签署文件之后，可在系统文件页面查看更多的文件信息、可进行下载、删除等操作。移动端和电脑端的核心页面实现效果如下图 4-17 和图 4-18 所示：

图 4-17 移动端文件页面和签署页面实现效果图

Fig.4-17 Mobile terminal file page and signature page to achieve effect map

图 4-18 电脑端文件页面实现效果图

Fig.4-18 The effect diagram of the computer end file page

4.4 负载均衡方案在系统中的应用与实践

Node.js 自身可作为服务器进行驱动，实际生产环境中运行 Node.js，其对文件的处理能力在性能上有一定的限制，因此在实际的开发环境中，常采用 Nginx 来处理静态的资源以及作为反向代理，从而解决了 Node.js 分布式以及负载均衡的相关问题。要在电子签名系统实现前端负载均衡，即在 web 服务层进行横向水平扩展，即增加若干 NodeJS 前端服务器实例节点，让原先一台服务器承担的压力分到多台 NodeJS 服务器，分别承担相同的压力，从而解决前端服务器性能的压力，配置的时候添加一个负载均衡器 Nignx 来对外提供统一的入口，然后将前端接收到的各个终端的请求按负载均衡策略的算法分发到 NodeJS 节点上进行相应的处理。该负载均衡方案设计使前端服务器具备接收大量数据请求的能力，并且某台 NodeJS 服务器宕机后，仍然能对外提供服务。同理，后端负载均衡是一样的原理。本文研究的目标是将后端的页面访问压力转移到前端，因此本文负载均衡方案的研究工作重点是在前端，前端负载均衡的设计图如下图 4-19 所示：

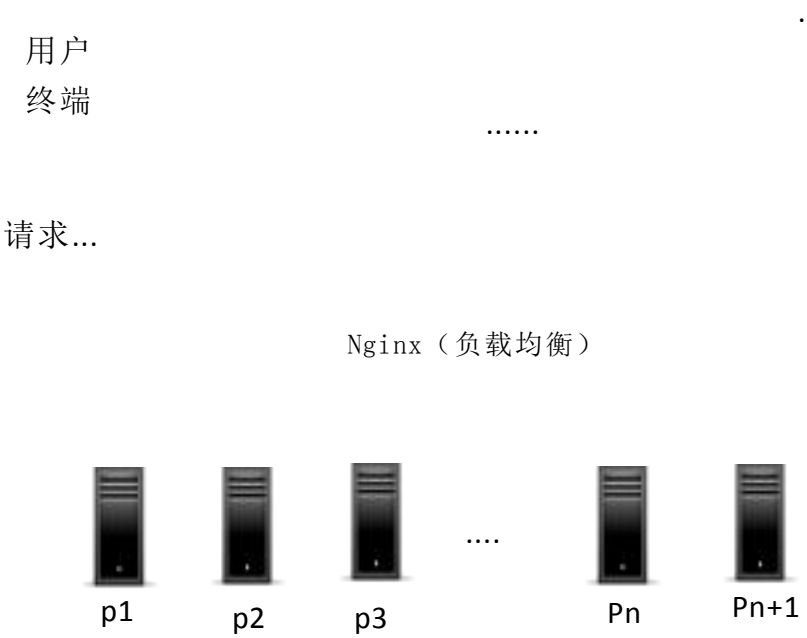

图 4-19 前端负载均衡的设计图

Fig.4-19 A design diagram for the load balance of the front end

用 Nignx 作为 Node.js 的负载均衡方案，对 Nignx.config 文件进行配置，配置核心代码如下所示：

```
upstream test {
        server 127.0.0.1:3000;
        server 127.0.0.1:3001;
        ......
}
server {
        listen 80;
```

```
        ......
        server_name 127.0.0.1;
        ......
        location {
            ......
            proxy_pass http://test;
        }
    }
```

配置的核心参数的释义为：配置了端口 3000 和 3001 各对应一个 Node.js 服务器，并且两个服务器分担着相同的工作，减轻单个服务器的压力，进而实现快速响应用户请求；在 upstream 模块上配置了两个 Node.js 服务器；最后代码“proxy_pass http://test”配置表示设置 proxy_pass http://test 做 HTTP 请求代理。

服务器这块的内容，在实际生产环境中，由运维人员来进行配置等操作，在工程的部署发布中进行实施。

前后端分离之后，前端和后端各自维护、打包工程，本项目中的部署方案采用将 Java 后端服务、Node.js 服务分别部署在不同的机器上，开发人员打包工程文件并通过 Docker(Docker 是一个开源的引擎，可为任何应用创建一个轻量的、可移植的、自给自足的容器，常用于 web 应用的自动化打包和发布)实现部署。并且，Node.js 服务和后端服务是在内网进行通讯，保证了服务器的安全和更优的性能。综上实践，从开发到部署整个流程，前端和后端实现了彻底分离，为后期维护和产品迭代提供了便利性。

4.5 基于 SaaS 应用的 web 性能优化模型实践总结

通过将 web 前端性能优化框架模型应用到实际 SaaS 系统中,具体实践包括基于 Node.js 的前后端分离模型、面向企业业务应用场景的组件化开发模型以及基于 Nginx 的负载均衡方案，在实践中遇到的问题和收获的经验总结如下：

(1) 解耦前后端职责，从而解决前后端协作问题：电子签名系统采用开发模型中的基于 Node.js 的前后端分离方案，即将前端和后端直接引入 Node.js 作为中间服务层，Node.js 服务器统一归入前端工程中，相对系

统之前的前端，本文叫“大前端”，分离之后，后端与前端只通过 restful api 接口进行连接，前端的路由配置等不在受限于后端，后端对外只需要关心提供的接口服务，将更多的精力放在后端服务的架构和性能上，脱离了前端的页面管理工作。

(2) 前后端开发由串行变为并行，提高整体开发效率：前后端分离之后，前后端通过接口进行直接联调，即使后端接口没有实现也不影响前端开发继续进行，前端可以通过调用 mock 数据先进行接口对接工作，同时后台进行接口的实际开发并进行单独的接口测试。前端完成之后，测试也可以按序进行。对于前端来说，后台接口开发完成之后，只是更换一下接口地址即可，从而实现前后端开发并行的效果。

(3) 前端开发人员按层分配任务，减小新人上手难度：前端开发人员技能层次不一，有的擅长 HTML 和 CSS，有的擅长 JavaScript 与后台接口对接，能力水平初级的只能写 UI 界面，但是不擅长与后台对接接口和联调难度大的任务，能力强的需要从静态界面到接口到联调全部完成，且任务繁重，在真正开发中，人力和开发安排不匹配，前端技术能力强的人是少数，而大部分人处于中级和初级水平，特别是新人，造成新人上手难度高。电子签名系统采用面向企业业务组件化的开发模型进行工程构建，轻易实现将前端开发人员进行分层，擅长网络层和 javascript 开发的人员，即可专注 web server 端与后台接口的对接，并且可专注书写提供客户端调用的 javascript 模块，而擅长 css 和 html 的前端开发人员，即可专注 UI 业务组件的实现，并可实现开发人员以组件划分任务，只需要沟通好，则可在项目开发周期中并行独立开发，互不影响，新人也可以快速上手，同时也可减少团队招人的困难。

(4) 实现多终端匹配，让各终端视觉效果和交互设计达到最优：前后端分离，前端基于 Node.js 可按照业务需求实现多终端匹配，并且通过面向企业业务应用场景的组件化开发模型，让各终端的体验效果和页面视觉效果达到最佳。

(5) 提高前端网页响应速度，让前端性能优化不受限于后端：电子签名系统还要解决的一个问题即是性能问题，最关键的是网页响应速度的问题。前后端分离之后，前端页面的部分性能优化不再受限于后端，让原先页面的反馈路径从后端服务直接转移到 Node.js 服务端，缩短了访问响应时间，从而提升了系统对用户的响应速度。并且，通过引入基于 Nginx 的负载均衡方案，前端服务端可根据用户访问量的情况扩展服务器，以

达到良好的访问速度。

综上，基于web前端的性能优化模型在解决电子签名系统的性能问题上进一步提高了开发效率，并且为企业级web应用程序的大型分布式架构、弹性计算架构、微服务架构、多端化服务打下坚实的基础。

4.6 本章小结

本章主要将web前端性能优化模型及相关技术应用到实际的SaaS级的企业web应用中，通过实际开发和分析，详细介绍了基于前后端分离模型在电子签名系统架构中的实现，并在实际应用中，可以发现该技术架构模型使得前后端工作职责更加清晰；前端应用组件化开发技术，大大提升了前端的开发效率，实现代码复用；基于Nginx的负载均衡方案实现了前端页面的负载均衡，解决了前端的性能问题。下一章将对web前端性能优化模型在电子签名系统中的实施进行全面的测试，从开发效率、性能等方面对电子签名系统优化前后的效果进行详细的对比分析。

5 测试和对比分析

本章对电子签名系统分别进行功能、性能测试，性能测试通过测试电子签名系统使用web前端性能优化框架模型前后，页面访问的响应速度来对比分析性能情况，并从开发效率、系统性能等方面对比进行分析，从而证明了本文提出的开发模型在SaaS应用开发过程中的有效性和高效性。测试环境如下表5-1所示：

表5-1 测试环境

Table 5-1 Testing environment

属性	具体环境及配置
硬件环境	CPU：Intel(R)Core(TM)i7-4790CPU@3.60GHz 四核八线程, 内存：8GB 1600MHz DDR3
操作系统	Ubuntu 14.04.1 LTS
web服务器	NodeJS 7.9.0
应用服务器	Tomcat 8.5.16
数据库	MysSql 5.6.17，MongoDB 3.6.2
性能测试工具	ApacheBench 2.3
编译环境JDK	JDK 1.8.0_131
浏览器	Chrome 64

5.1 功能测试

针对该电子签名系统，功能性测试主要从系统实现的业务逻辑、系统功能实现程度以及不同终端网页视觉渲染效果三方面做测试，该系统面向多终端设计，功能测试分别对移动端和电脑端做了测试，电脑端的功能模块包括登录、注册、文件、文件签署、文档预览、签章管理、个人信息设置、实名认证，移动端的功能模块包括文件、文件签署、个人信息，根据系统功能进行可测试用例编写并根据测试用例进行了测试，发现的bug已经得到及时的解决，如下表5-2是部分测试用例和测试结果：

表 5-2 部分测试用例和测试结果

Table 5-2 Some test cases and test results

编号	模块名称	功能点	操作步骤	预期结果	实测结果(F/T)
login_001	登录	登录完备性测试	1.输入系统不存在的邮箱号码 2.任意输入密码 3.点击登录	提示用户不存在	T
login_002			1.输入系统未激活的邮箱号码 2.任意输入密码 3.点击登录	提示“账户未激活，立即激活？”	F:未提示
register_001	注册	注册完备性检查	1.注册流程中，将手机号或邮箱输入，姓名输入、密码、确认密码和验证码中任意一个或多个格式填写错误或不填	点击立即注册无法完成注册	T
register_002		手机号或邮箱系统已存在	1.手机号码填写系统已存在的用户 2.其他输入按照正确的格式填写 3.点击“立即注册”	提示“用户已存在”	T
doc_001	文件操作	文件删除	1.点击“全部文件” 2.点击某条信息的“删除按钮” 3.查看页面	该条信息已被删除	T
doc_002		已完成文件预览	1.点击“已完成” 2.查看某条信息“预览按钮” 3.查看新页面	显示该文件详细内容	T

doc_003	已完成文件文档下载	1. 点击“已完成” 2. 查看某条信息“下载”按钮，选择“文档下载” 3. 查看新页面	弹出下载页面并开始下载	T
doc_004	草稿编辑按钮	1. 点击“草稿” 2. 点击某条信息的“编辑”按钮 3. 查看页面	跳转到添加文档页面	T

视觉渲染效果测试主要是测试移动端和电脑端网页的渲染是否正常，按照不同终端对应渲染的指定页面，使用 Chrome 自带的终端模拟工具进行测试，移动端和电脑端的网页测试结果显示是正常，页面根据不同终端进行了适配渲染。如下图 5-1 以登录界面分别在移动端和电脑端的渲染测试效果图作为示例：

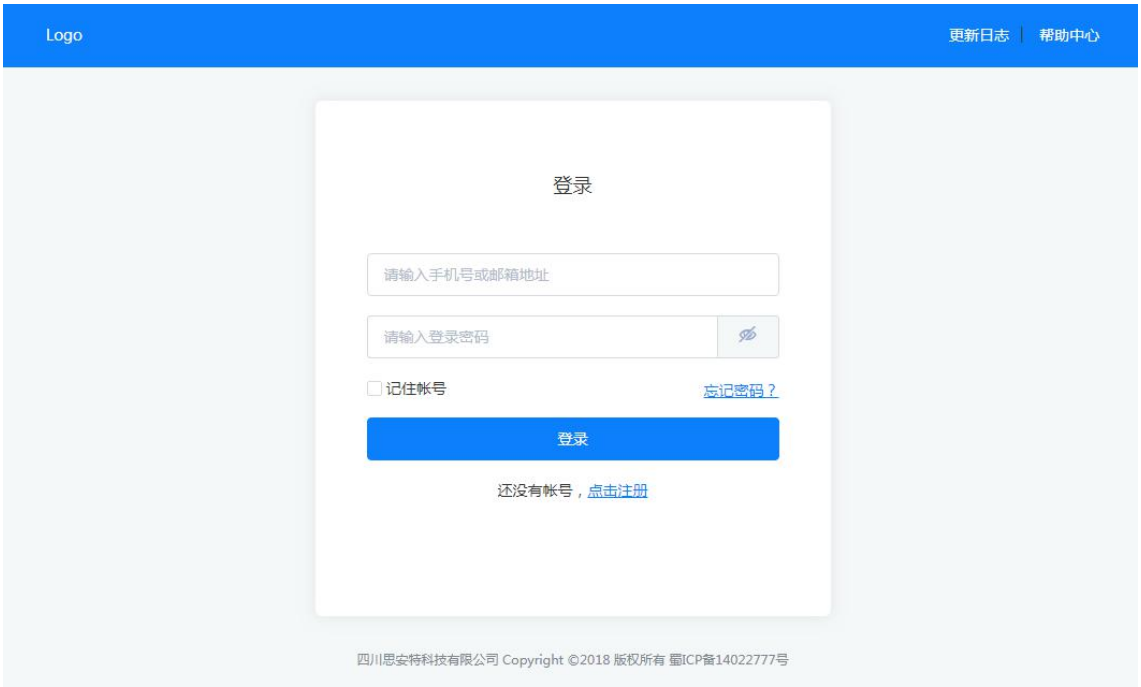

图 5-1 登录页面在移动端和电脑端自动适配的测试效果图

Fig. 5-1 Test results of automatically matching login pages at mobile terminals and computer terminals

5.2 性能测试

使用压力测试工具 ab（ab 是 apache 自带的压力测试工具），分别模拟 4 组数据，并发请求访问本地发布的 2 种单服务器环境的相同首页，测试电脑端前后端分离前后的页面访问响应的速度，测试详情和结果如下：

(1) 第一组测试数据：通过测试工具 ab 模拟 1000 个人的 1000 个并发请求，可以明显看出采用前后端分离之后，页面响应速度平均提升约 4.4 倍。测试详情如下表 5-3 所示：

表 5-3 第一组测试数据

Table 5-3 First set of test data

模式	主页地址	请求数量(个)	平均响应耗时(ms)	请求成功(个)	请求失败(个)
模拟 1000 个人的 1000 个并发请求					
分离前	home	1000	9.34ms	1000	0
分离后	home	1000	2.11ms	1000	0

(2) 第二组测试数据：通过测试工具 ab 模拟 5000 个人的 5000 个并发请求，可以明显看出采用前后端分离之后，页面响应速度平均提升约 4.1 倍。测试详情如下表 5-4 所示：

表 5-4 第二组测试数据

Table 5-4 Second sets of test data

模式	主页地址	请求数量(个)	平均响应耗时(ms)	请求成功(个)	请求失败(个)
模拟 5000 个人的 5000 个并发请求					
分离前	home	5000	10.42ms	5000	0
分离后	home	5000	2.53ms	5000	0

(3) 第三组测试数据:通过测试工具 ab 模拟 10000 个人的 10000 个并发请求，可以明显看出采用前后端分离之后，页面响应速度平均提升约 4.9 倍。测试详情如下 5-5 表所示：

表 5-5 第三组测试数据

Table 5-5 Third sets of test data

模式	主页地址	请求数量(个)	平均响应耗时(ms)	请求成功(个)	请求失败(个)
模拟 10000 个人的 10000 个并发请求					
分离前	home	10000	10.09ms	10000	0
分离后	home	10000	2.07ms	10000	0

(4) 第四组测试数据:通过测试工具 ab 模拟 50000 个人的 50000 个并发请求，可以明显看出采用前后端分离之后，页面响应速度平均提升约 5.7 倍。

测试详情如下 5-6 表所示：

表 5-6 第四组测试数据

Table 5-6 Fourth sets of test data

模式	主页地址	请求数量(个)	平均响应耗时(ms)	请求成功(个)	请求失败(个)
模拟 50000 个人的 50000 个并发请求					
分离前	home	50000	10.26ms	50000	0
分离后	home	50000	1.8ms	50000	0

通过以上测试数据，可以看出，相比较传统前后端 MVC 的开发模式，采用前后端分离模型之后网页响应速度显著提升，且并发请求量越高，分离后的性能优势越明显。

5.3 模型优化前后分析

经过前后端分离，原先开发 Java 后台的人员只需要专注 API 的设计和实现，无需关心前端页面路由，前后端的工作职责和界限更加清晰和明确；前端脱离后端之后，页面性能的优化不在受限于后端的服务，可以最大程度的优化页面响应速度和性能；前端以组件化的开发思想进行工程构建，可以轻易将前端开发人员进行分层，擅长网络层和 javascript 开发的人员，即可专注 web server 端与后台接口的对接，并且可专注提供客户端调用的 javascript 模块，而擅长 css 和 html 的前端开发人员，即可专注 UI 业务组件的实现。最后，可实现开发人员以组件划分任务，只需要沟通好，则可在项目开发周期中并行独立开发，互不影响。

(1) 前后端开发模式由串行变成了并行，开发效率明显提升，前端和后端的工作没有了耦合，职责界限清晰，现在只需要通过接口进行对接，即使后端没有及时提供接口，前端也不用等着后端人员，可以通过 mock 数据进行模拟调用，从而完成前端的功能，后台接口开发完成，前端只是需要修改接口地址即可。

(2) 前端采用面向企业业务组件化的开发思想，让前端任务模块化，人员任务以组件划分，独立开发，减小各自之间的依赖，从而提高开发效率，随着应用的逐渐扩大和复杂化，组件化的方式让工程维护变得简单和轻

松。

(3) 前后端分离之后，前端页面的部分性能优化不再受限于后端，让原先页面的反馈路径从后端服务直接转移到 Node.js 服务端，缩短了访问响应时间，从提升了系统对用户的响应速度。于此同时，静态页面的访问可以通过引用 Nignx，同时也让后台不再关心静态页面访问带来的压力。并且前端服务端可根据用户访问量的情况扩展服务器，以达到良好的访问速度。

(4) 使用压力测试工具 ab（ab 是 apache 自带的压力测试工具），分别模拟 4 组数据，并发请求访问本地发布的 2 种单服务器环境的相同首页访问效果，采用前后端分离模型之后性能显著提升，且并发请求量越高，分离后的性能优势越明显。

5.4 本章小结

经过对本文提出的 web 前端性能优化模型的实践和测试，证明该模型在 SaaS 应用开发过程中的可用性和高效性。该模型从开发模式上让前后端彻底分离、解耦前后端工作，让前端和后端相互独立，使得网页性能的优化不在受限于后端服务能力，大大提升了网页的响应速度，从而增强企业竞争力。

6 总结与展望

6.1 总结

经过对本文提出的web前端性能优化模型的实践和测试，证明了该模型在SaaS应用开发过程中的可用性和高效性，在SaaS应用性能问题上进一步提高了开发效率，并且为企业级web应用程序的大型分布式架构、弹性计算架构、微服务架构、多端化服务打下坚实的基础，从而增强企业竞争力。其带来的具体好处表现在如下几方面：

(1) 解耦前后端职责，从而解决前后端协作问题：基于Node.js的前后端分离方案，后端与前端只需要通过restful api接口进行连接，前端的路由配置等不再受限于后端，后端对外只需要关心提供的接口服务，将更多的精力放在后端服务的架构和性能上，脱离了前端的页面管理工作。

(2) 前后端开发由串行变为并行，提高整体开发效率：前后端分离之后，前后端通过接口进行直接联调，即使后端接口没有实现也不影响前端开发继续进行，前端可以通过调用mock数据先进行接口对接工作，同时后台进行接口的实际开发并进行单独的接口测试。前端完成之后，测试也可以按序进行。对于前端来说，后台接口开发完成之后，只是更换一下接口地址即可，从而实现前后端开发并行的效果。

(3) 前端开发人员按层分配任务，减小新人上手难度：面向企业业务组件化的开发模型进行工程构建，可以轻易实现将前端开发人员进行分层，擅长网络层和javascript开发的人员，即可专注web server端与后台接口的对接，并且可专注书写提供客户端调用的javascript模块，而擅长css和html的前端开发人员，即可专注UI业务组件的实现，同时新人也可以快速上手，减少团队招人的困难。

(4) 实现多终端匹配，让各终端视觉效果和交互设计达到最优：前后端分离，前端基于Node.js可按照业务需求实现多终端匹配，并且通过面向企业业务应用场景的组件化开发模型，让各终端的体验效果和页面视觉效果达到最佳。

(5) 提高前端网页响应速度，让前端性能优化不受限于后端：前后端分离之后，前端页面的部分性能优化不再受限于后端，让原先页面的反馈路径从后端服务直接转移到Node.js服务端，缩短了访问响应时间，提升了系统对用户的响应速度。并且，通过引入基于Nginx的负载均衡方案，

前端服务端可根据用户访问量的情况扩展服务器，以达到良好的访问速度。

6.2 展望

本文提出的web前端性能优化模型主要面向企业大型web应用，且当前应用是一款需要持续迭代的产品，不是所有网站开发都是适合的，比如网站门户的开发，其应用本身比较简单，没必要采用这种开发模型，如果使用该开发模型，反而增加开发难度和增大维护成本。另外一点，前后端分离之后，对于前端开发人员的要求也增大了,前端技术要求增大是必然的,企业要提升竞争力，如果企业技术跟不上技术的发展，将会面临淘汰。传统的开发模式下，服务端的事务交由后端人员负责，前后端分离之后，Node.js服务器归并到前端人员负责，因此，需要前端至少有一个人员负责Node.js服务端的处理、页面路由配置和维护、发布等事项。前后端分离之后，虽然前端和后端只需要通过接口进行对接，职责分明，但是与此同时，对前后端开发人员的交流能力也提出了更高的要求。

致 谢

在黄晓芳老师的理论指导和自己的技术积累下，经过大量的文献阅读和理论学习，并在前端技术方面进行学习和应用，从项目实践和研究中，找到基于 web 前端性能优化方面的研究内容，并通过研究和实践证明了研究内容的有效性和实际意义，完成了毕业设计所要求的内容。在此过程中，我不只是完成了一个毕业设计，更是自我能力提升和自我升华的一次历练，总结如下：

(1) 写作能力得到提升：由于我自身比较偏工程，对于学术研究，学习的方式方法，一开始没有头绪，对于如何很好的写论文，把自己的观点和内容如何有条理的表达出来，给我带来极大的挑战，这期间，导师黄晓芳给了我很多的指导，我在完成毕业设计的过程中，不仅学会了如何去思考问题，并且在写文方面积累了一些经验，对我以后的工作起到很大的作用。

(2) 勇敢挑战自己，不为人生设限：我常常在想读研有什么意义，难道只是为了拿一个证，只是为了找更高工资的工作，然而不是，这些都没有意义。人这一生应该是在追求自我完善，为获得自我升华而快乐的一个过程，而不是金钱、名利，当初选择读研，也是因为我不知道自己在追寻什么，想要通过读研找寻我人生意义和目标。很幸运，在这个实验室中，这个平台给了我更多发挥自我的空间，并在导师和大家的鼓励与帮助之下，让我捡起勇气去寻找另一个自己，而不是将自己设限于现状，我从自卑变得自信，从自闭变得开朗，充满正能力，让我更好的了解了自己并接纳自己，并且敢于改变自己。这真的是一种重生的感觉。

(3) 思考力得到提升：如今我们处在一个互联网时代，这是一个知识共享、科技技术飞速发展的年代，发现问题、分析问题和解决问题的能力可以看出一个优秀的人和一个平庸者的区别，学习能力是保证一个人与时俱进的基础，独立思考能力是一个人在这个信息爆发中不人云亦云的护身符，脚踏实地是一个人能长久立足于社会的宝贵品质，坚持不懈是一个人成功做成一件事的必备精神。这些宝贵的认识将伴随我一生。

总的来说，与其说这是在做一次毕业设计，不如说是我人生道路中的一次转折点，让我的心智得到提升并且找到那个真正的自己。这对于我以后的工作学习都是一笔宝贵的财富。

首先，向我的导师黄晓芳老师致以崇高的敬意和衷心的感谢。黄晓芳老师总是以严谨的治学理念和认真负责的工作态度，给予了我莫大的教诲和影响。然后，感谢同学朋友们对我毕业设计的帮助，正是他们无时无刻的帮助，使我拓宽了知识面，并给了我一个温暖的学习环境，学习到了更多的东西。

参考文献

[1]（美）桑德斯著.高性能网站建设指南 前端工程师技能精髓[M].2015
[2]［美］桑德斯著，高性能网站建设进阶指南，电子工业出版社，2010.
[3] 孔令旭. 基于 Node.js 的前后端分离框架的实现与应用[D]. 华中科技大学, 2016.
[4] 吴贺. 前后端解耦模式及开发[J]. 计算机系统应用, 2017, 26(2):217-221.
[5] 仇晶, 黄岩, 柴瑜晗. 基于 Node.js 中间层 Web 开发的研究与实现——以微信图书借阅平台为例[J]. 河北工业科技, 2017, 34(2):118-124.
[6] Douglas, Christopher, Wilson. Express 4.x API 中文手册[EB/OL]. http://www.expressjs.com.cn/4x/api.html, 2018
[7] Johannes,Ewald.webpack 中文文档[EB/OL]. https://www.webpackjs.com/concepts/, 2018
[8] Yuxi, (Evan), You. Vue.js API[EB/OL]. https://vuefe.cn/v2/api/, 2018
[9] 凡涛涛. 一种基于 Nginx 的动态负载均衡策略[D]. 中国科学院大学, 2016.
[10] 张尧. 基于 Nginx 高并发 Web 服务器的改进与实现[D]. 吉林大学, 2016.
[11] 杨小娇. 轻量级高并发 Web 服务器的研究与实现[D]. 南京邮电大学, 2014.
[12] 王永辉. 基于 Nginx 高性能 Web 服务器性能优化与负载均衡的改进与实现[D]. 电子科技大学, 2015.
[13] 李曰斌，詹舒波.基于 Vue 的前端组件化研究与实践[Z].2016.09.09
[14] 周兴宇，卞佳丽. 基于 React 的前端组件化研究与设计[EB/OL].北京：中国科技论文在线 [2016-01-05].
[15] 黄思远，詹舒波. 前端组件化思想及实践[EB/OL].北京：中国科技论文在线 [2015-12-23].
[16] banri. elementUI[EB/OL]. http://element.eleme.io/#/zh-CN，2017
[17] 李晓峰.Web 工程前端性能优化[J].电子科技,2015,第 28 卷(第 5 期): 184-186
[16] 刘柳.基于 Web 前端的性能优化方案研究[D].华中科技大学,2015
[19] 王成，李少元，郑黎晓，缑锦，曾梅琴，刘慧敏.Web 前端性能优化方案与实践[J].计算机应用与软件,2014,第 31 卷(第 12 期): 89-95，147
[20] 陈秋实.移动 Web 前端性能优化方法及其应用研究[D].华中科技大学,2015

[21] 姚昌.SaaS平台前端性能优化的研究与实现[D].北京邮电大学,2015

[22] 陈俊璘.Web应用程序的性能优化研究[D].电子科技大学,2014

[23] 黄静, 李炳. 基于Nginx的Web服务器性能优化研究[J]. 浙江理工大学学报, 2016, 35(4):600-606.

[24] 周萌, 姜增辉, 吴际达,等. 响应式快速开发框架研究与实现[J]. 电子科学技术(北京), 2016, 3(4):450-453.

[25] 彭玲玲,李诗莹,冯爽.基于前端的Web性能优化[J].收藏,2017,24:057.

[26] TJ, Holowaychuk. Node.js v8.11.1 文档[EB/OL]. http://nodejs.cn/api/ ,2018

攻读学位期间发表的与学位论文内容相关的学术论文及研究成果

杜艳美,黄晓芳. 面向企业级web应用的前后端分离开发模式及实践.西南科技大学学报（自然科学版）,2018,33(2):88-93.