四川农业大学

本科毕业论文（设计）

（2018 届）

|  |  |
| --- | --- |
| 题 目： | 基于node.js的文档框架的研究与实现 |
| 学 院： | 信息工程学院 |
| 专 业： | 计算机科学与技术 |
| 学生姓名： | 许娜 学号： 20158867 |
| 导师： | 潘勇浩 职称： 讲师 |
| 完成日期： | 2019 年 3月 25 日 |

目 录

1 绪论 2

1.1 选题背景 2

1.2 研究现状 2

2 需求分析 2

2.1 项目需求 2

2.2 需求的提出 3

3 项目可行性分析 4

3.1 项目可行性 4

3.2 技术可行性 4

4 技术实现路线 4

4.1 相关理论知识 4

4.2 相关技术知识 5

4.3 开发环境 5

4.4 Less预处理语言 5

4.5 Express框架 6

5 总体设计 7

5.1 系统总体设计 7

5.2 框架流程控制 7

6 详细设计和实现 8

6.1 开发环境及工具详情 8

6.2 框架模块的实现 8

6.2.1 框架模块概述 8

6.2.2 具体实现 9

6.3 服务器端的实现 11

6.3.1 Express框架概述 11

6.3.2 具体实现 11

7 程序测试 17

7.1 界面测试 17

8 结论 23

参考文献 24

致谢 25

基于node.js的文档框架的研究与实现

计算机科学与技术专业 许娜

导师 潘勇浩

**摘要**： 随着开源活动的不断发展，全世界越来越多的开发者加入了开源程序的潮流中。截止2018年，著名的开源社区网站Github已经拥有2800万用户以及2800万个开源项目[1]。对于开源项目的开发者来说，文档可以方便协同开发者和使用者更好的了解项目及使用它们。本文将从前台技术（HTML[2]，CSS[3]，JavaScript[4]）、后台技术（Node.js[5]）等方面来论述，如何通过简单的配置来帮助开发者们生成一个文档网站，并在服务器中运行。

**关键词**： NodeJS；Express；开源文档

**Research and Implementation of Document Framework Based**

**on Node.js**

Computer Science and Technology Na Xu

Directed by YongHao Pan

**Abstract**: With the continuous development of open source activities, more and more developers around the world have joined the trend of open source programs. By 2018, the famous open source community website Github had 28 million users and 28 million open source projects. For developers of open source projects, documents can facilitate collaborative developers and users to better understand and use projects. This article will discuss from the front-end technology (HTML, CSS, JavaScript), background technology (Node. js) and other aspects, how to help developers generate a document website through simple configuration, and run in the server.

**Key words**: NodeJS; Express; Open source document

1 绪论

1.1 选题背景

文档是软件开发使用和维护中的必备资料。

软件文档或者源代码文档是指与软件系统及其软件工程过程有关联的文本实体。文档的类型包括软件需求文档，设计文档，测试文档，用户手册等。其中的需求文档，设计文档和测试文档一般是在软件开发过程中由开发者写就的，而用户手册等非过程类文档是由专门的非技术类写作人员写就的。

文档能提高软件开发的效率，保证软件的质量，而且在软件的使用过程中有指导、帮助、解惑的作用，尤其在维护工作中，文档是不可或缺的资料。

在GitHub等开源社区中有非常多的开源项目，它们需要把文档公布出来给使用者及其他开发者查阅，对于一些大型的项目，文档数量众多且篇幅较长，甚至需要一个独立的网站来承载这些文档，方便其他开发者查阅。对于某些开发者来说，或许他没有开发过网站，或者没有必要在文档展示方面花费高昂的时间成本，这时候我们就需要一个方便快捷的框架来帮助文档维护者快速建立起一个可访问的文档网站。

1.2 研究现状

国内外对文档框架的建设和研究并不多，一方面是对于一个普通的开发者来说，他需要维护的文档可能并不多，所以普通开发者搭建框架的意识比较薄弱。另一方面，在各个公司的内部可能维护着比较规范的文档框架及流程，但很不幸，对于公司来说，内部的规范和流程是不会开源给非公司内部的人员使用的。

不过在开源社区GitHub中，还是能找到一些API文档框架，下面是它们的思路及解决方案。

（1）国内文档框架ShowDoc的解决方案

ShowDoc[6]是一款基于PHP并针对于团队的API文档框架。ShowDoc采用markdown编辑器，并将其存放在数据库中，在展示时将其解析为HTML展现。并提供许多快捷编辑的功能，便于开发者来写作。但它的功能注重于团队共享，方便团队沟通，对于个人使用者或独立项目并不是十分合适。且其数据的存储基于mysql数据库，需要用户有一定的维护数据库的经验，部署和维护起来并不是很方便。

（2）国外框架HEXO的解决方案

Hexo 是一个快速、简洁且高效的博客框架。Hexo 使用 Markdown（或其他渲染引擎）解析文章，在几秒内，即可利用node.js生成静态网页。

Hexo以文件作为数据，利用文件生成相应的静态网页并使用node.js提供的HTTP服务器对静态文件提供访问，对于访问者来说，使用静态文件作为数据源省去了数据库查询以及数据往返的时间，它的访问速度无疑要优于使用数据库的文档网站。

根据上面的比较可以看出，Hexo的解决方案更加优秀，在速度和个性化上，Hexo的优势比较明显，所以这次研究会以Hexo的设计思路作为参考。

2 需求分析

2.1 项目需求

本项目旨在为程序员用户提供一个高效，便捷的文档框架，用以快速搭建出一个静态网站，并开启服务器对静态文件提供网络访问。具体包括以下内容：

（1）高效实时处理，对markdown文件进行快速解析，并生成对应的静态文件。

（2）主题自定义：框架提供多种主题，用户通过配置文件，可以方便的改动网站的样式主题，网站更加多样化。同时提供主题接口，专业用户可以通过该接口高度自定义网站的显示样式。

（3）功能模块可扩展：对网站提供多种工具，用户通过配置文件决定使用哪些功能模块，并且提供命令行工具对网站进行多种控制。

（4）快速部署：根据HTTP服务器以及node.js的工作特点，只需要一条命令就可以对网站进行快速部署，不需要繁琐的配置就可对网站提供快速访问。

（5）网站配置简单：用户不需要编写代码，只需要在配置文件中填写配置项即可完成配置。

3 项目可行性分析

3.1 项目可行性

开源社区的用户和开源项目逐渐增多，对开源文档的需求当然也是日益上升，所以，一个开源文档框架在当今开源社区是可行的，可适用的，可推广的。

3.2 技术可行性

本框架使用JavaScript作为主要的开发语言。以基于Node.js（Node.js 是一个基于 Chrome V8 引擎的 JavaScript 运行环境，使用了一个事件驱动、非阻塞式 I/O 的模型，使其既轻量级又有效。其中的npm包管理平台更是为nodeJS的操作带来了巨大的方便性，本项目就可以使用npm进行快捷的安装，并且项目中使用的cheerio模块也是使用npm安装的）的MVC框架Express作为服务器的开发。NodeJS 事实上既是一个运行时环境，同时又是一个库。其适合的适用场景为应用提供高可用的场景，有研究表明运行NodeJS的服务器能支持数万个并发连接，这 差不多是PHP语言编写的服务器程序处理能力的两倍[7]。故，在技术层面来说是可行的。

4 关键技术理论

项目所用技术包括：HTML、CSS、JavaScript、Less（CSS预处理语言）、NodeJS、JSON、Express（基于NodeJS的MVC框架），其中的npm包管理平台更是为nodeJS的操作带来了巨大的方便性）的MVC框架Express作为服务器的开发。

4.1 相关理论知识

（1）markdown原理与应用

（2）node.js相关应用

（3）JSON应用

（2）数据结构

（3）超文本传输协议（HTTP，HyperText Transfer Protocol）

4.2 相关技术知识

（1）JavaScript程序开发

（2）ECMAScript6.0 相关编程知识和技术

（3）Express（基于nodeJS的MVC框架）开发技术

4.3 开发环境

（1）操作系统：Windows10

（2）开发工具：Visual Studio Code

（3）浏览器：Chrome浏览器、Safari浏览器、Edge浏览器

（4）测试工具：Chrome浏览器、node运行环境

4.4 Markdown标记语言

Markdown是一种轻量级标记语言，具有纯文本格式语法。它的设计允许它转换为许多输出格式，但同名的原始工具只支持HTML。Markdown通常用于格式化自述文件，在在线论坛中编写消息，以及使用纯文本编辑器创建富文本。 由于Markdown的初始描述包含含糊不清和未回答的问题，因此多年来出现的实现存在细微的差异，而且许多实现了语法扩展[8]。

本项目使用markdown-it将Markdown翻译为HTML语言。

4.5 Less预处理语言

CSS（层叠样式表）是一门历史悠久的标记性语言，同 HTML 一道，被广泛应用于万维网（World Wide Web）中。HTML 主要负责文档结构的定义，CSS 负责文档表现形式或样式的定义。

作为一门标记性语言，CSS 的语法相对简单，对使用者的要求较低，但同时也带来一些问题：CSS 需要书写大量看似没有逻辑的代码，不方便维护及扩展，不利于复用，尤其对于非前端开发工程师来讲，往往会因为缺少 CSS 编写经验而很难写出组织良好且易于维护的 CSS 代码，造成这些困难的很大原因源于 CSS 是一门非程序式语言，没有变量、函数、SCOPE（作用域）等概念。LESS 为 Web 开发者带来了福音，它在 CSS 的语法基础之上，引入了变量，Mixin（混入），运算以及函数等功能，大大简化了 CSS 的编写，并且降低了 CSS 的维护成本，就像它的名称所说的那样，LESS 可以让我们用更少的代码做更多的事情。

4.6 Express框架

Express 是一个简洁而灵活的 node.js Web应用框架, 提供一系列强大特性帮助你创建各种Web应用。Express 不对 node.js 已有的特性进行二次抽象，我们只是在它之上扩展了Web应用所需的功能。丰富的HTTP工具以及来自Connect框架的中间件随取随用，创建强健、友好的API变得快速又简单

Express 是一个简洁而灵活的 node.js Web应用框架, 提供了一系列强大特性帮助你创建各种 Web 应用，和丰富的 HTTP 工具[9]。

使用 Express 可以快速地搭建一个完整功能的网站。

4.7 Cheerio工具

Cheerio是快速，灵活和精简的核心jQuery实现，专为服务器而设计的便捷工具。cheerio实现了核心jQuery的一个子集。 Cheerio从jQuery库中删除了所有DOM不一致和浏览器残骸，揭示了它真正华丽的API。Cheerio使用非常简单，一致的DOM模型。因此，解析，操作和渲染非常有效。初步的端到端基准测试表明，cheerio比JSDOM快约8倍。Cheerio包装了parse5解析器， Cheerio几乎可以解析任何HTML或XML文档。

使用Cheerio可以做快速的文档解析。

4.8 npm

npm 是 Node.js 官方提供的包管理工具，他已经成了 Node.js 包的标准发布平台，用于 Node.js 包的发布、传播、依赖控制。npm 提供了命令行工具，使你可以方便地下载、安装、升级、删除包，也可以让你作为开发者发布并维护包。

npm 是随同 Node.js 一起安装的包管理工具，能解决 Node.js 代码部署上的很多问题，常见的场景有以下几种：

* 允许用户从 npm 服务器下载别人编写的第三方包到本地使用。
* 允许用户从 npm 服务器下载并安装别人编写的命令行程序到本地使用。
* 允许用户将自己编写的包或命令行程序上传到 npm 服务器供别人使用。

npm 的背后，是基于 couchdb 的一个数据库，详细记录了每个包的信息，包括作者、版本、依赖、授权信息等。它的一个很重要的作用就是：将开发者从繁琐的包管理工作（版本、依赖等）中解放出来，更加专注于功能的开发。

本项目使用的Express以及Cheerio都有托管在npm上的版本，通过npm可以很方便的进行模块管理。Npm使用package.json文件对本地的模块进行管理，本项目同时通过git和npm进行托管，用户也可以非常方便的使用npm或是git来安装本项目。

本项目的package.json如下：

{

"name": "radoc",

"version": "1.0.0",

"description": "a document framework based on Node.js",

"main": "index.js",

"author": "rahsu",

"license": "ISC",

"dependencies": {

"cheerio": "^1.0.0-rc.2",

"express": "^4.16.2",

"less": "^3.9.0",

"markdown-it": "^8.4.0"

}

}

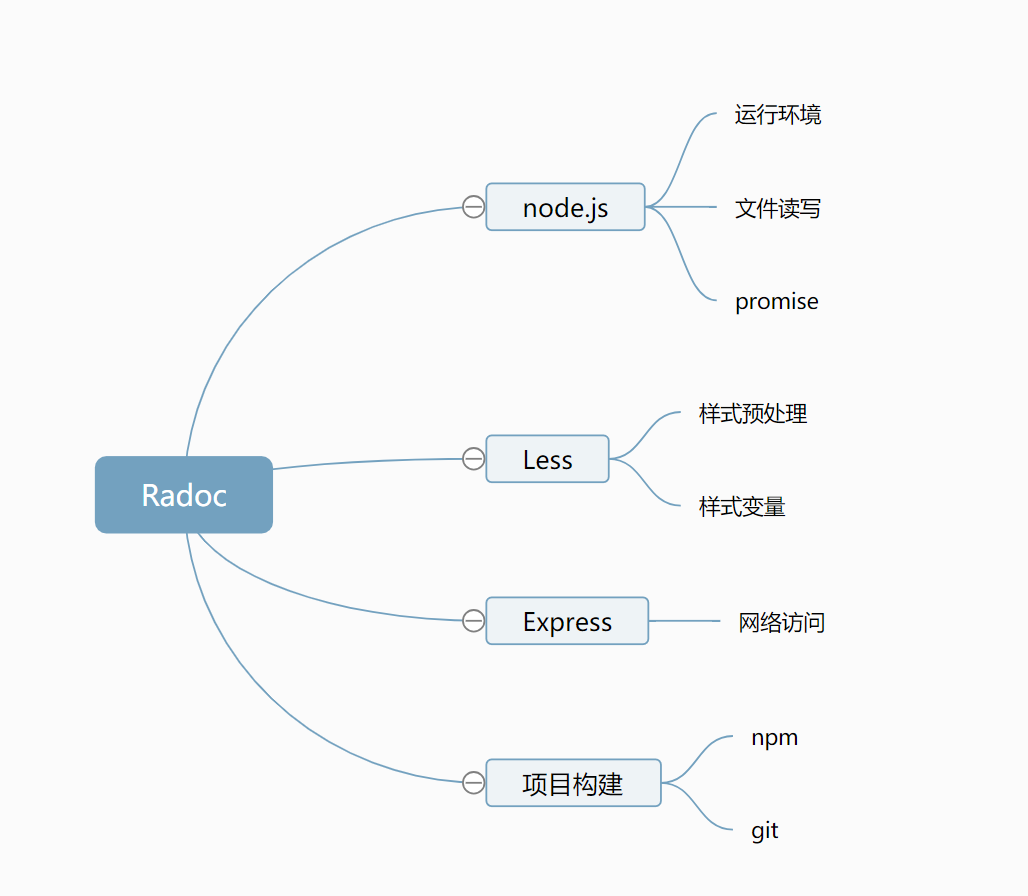


图1 技术架构图

5 总体设计

5.1 系统总体设计

基于node.js的文档框架通过node.js对源文件和配置文件进行解析，生成静态页面，并使用express框架对静态文件提供服务器访问。

用户在使用该框架时，将源文件放入指定的文件夹中，然后在配置文件中按照自己的需求配置样式和组件。

框架会根据配置文件，将用户指定的组件与由源文件解析而成的dom元素组合生成静态文件，然后由node.js服务器对这些静态文件提供访问。

5.2 框架流程控制

框架的运行流程如下图1所示：



图2 文档框架的运行流程图

主流程文件的代码如下：

const md = require("markdown-it")();

const fs = require("fs");

const re = require("./modules/re");

const wf = require("./modules/write-file");

const pa = require("./modules/parse");

const homeRender = require("./modules/home-render");

const catalogRender = require("./modules/catalog-render");

// 读取配置文件，解析站点配置

const prime\_config = require('./site-config.json');

const site\_config = pa.analysis\_config(prime\_config);

// 将预处理css与js输出到static文件夹

// css预处理：less输出到static中

// 读取theme\_color并成css

wf.generateThemeColor('./src/less/theme-color.less',site\_config.theme\_color);

// 将home-theme.less编译为css

wf.less('./src/less/home-theme.less','./static/css/home-theme.css');

// 将guide-theme.less编译成css

wf.less('./src/less/guide-theme.less','./static/css/guide-theme.css');

wf.less('./src/less/apollo.less','./static/css/apollo.css');

//将static静态文件复制到publish中(包括css,js,font和img)

wf.copyDir('./static/css','./publish/css');

wf.copyDir('./static/font','./publish/font');

wf.copyDir('./static/js','./publish/js');

wf.copyDir('./static/img','./publish/img');

//将post\_source中的img和html原样复制

wf.copyDir('./post\_source/post\_img','./publish/post\_img');

wf.copyHtml('./post\_source','./publish');

// 将post\_sourse中的md文件转换为html文件

wf.writeAllMd();

// 生成首页

homeRender.renderHome();

// 生成文档页

catalogRender.renderCatalog();

5.3 项目的数据设计

本项目需要用户在配置文件中对其做一定的配置，那么，合理地设计配置文件数据，才能让用户快速，简单的配置好网站。

本项目使用JSON这种数据格式来作为配置文件的配置格式。

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。易于人阅读和编写。同时也易于机器解析和生成，JSON采用完全独立于编程语言的文本格式来存储和表示数据。简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言，有效地提升网络传输效率。它基于JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999的一个子集。

本项目一共有两个配置文件需要用户进行配置，一个是关于网站的配置文件site-config.json，一个是关于首页的配置文件home-config.json。

1. site-config.json

site-config.json中包含对网站的配置，其形式比较简单，它表示为一个json的名称/值对，示例如下：

{

"title":"Radoc",

"theme":"apollo",

"code\_theme":"arduino-light",

"theme\_color":"#f00",

"catalog":[

{

"title":"基础",

"src":"test.html",

"children":[

{

"title":"安装",

"src":"fun fact about null.html",

"children":[

{

"title":"前端问题总结",

"src":"前端问题总结.html"

}

]

}

]

}

],

"first\_show":"你不知道的JSON.html"

}

不同的名称/值对表示对网站的一个配置。如title表示网站名称，theme表示网站主题，等等不一一列举。其中catalog为一个数组，表示网站的文档关系，它允许最多三层嵌套，在catalog-render.js中会按照这个配置将其渲染为三级目录。

1. home-config.json

home-config.json中包含对网站首页的配置，它整体为一个JSON名称/值对，其中的层级关系用名称/值对的嵌套来表示。

首页布局大致分为四个部分： header（首部导航），banner（标题横幅）， content（内容），footer（尾部信息）

header部分示例如下：

{

"title":"Radoc",

"header":{

"logo":[

{

"type":"img",

"src":"trecer.jpg"

},{

"type":"text",

"content":"Radoc"

}

],

"links":{

"position":"right",

"children":[

{

"type":"link",

"href":"#",

"text":"团队"

},{

"type":"dropdown",

"text":"多语言",

"children":[

{

"href":"http://www.baidu.com",

"text":"学习"

},{

"href":"www.baidu.com",

"text":"团队"

},{

"href":"www.baidu.com",

"text":"生态系统"

}

]

}

]

}

}

}

6 详细设计和实现

6.1 开发环境及工具详情

本设计在Windows10操作系统上开发完成，使用Visual Studio Code作为建立软件后台开发工程、编写程序代码的主要开发工具（Visual Studio Code是一个轻量但功能强大的源代码编辑器，可在桌面上运行，适用于Windows，macOS和Linux。它内置了对JavaScript，TypeScript和Node.js的支持，并具有丰富的其他语言（如C ++，C＃，Java，Python，PHP，Go）和运行时（如.NET和Unity）的扩展生态系统。）。且，Visual Studio Code能够扩展各种框架，都能够做到代码高亮，错误提示等，这是作为JavaScript系开发的一个必备的开发工具[10]。

由于考虑到项目中Web部分需要运行在不同的浏览器当中，因此选用比较有代表性的4款浏览器作为了运行环境。浏览器的内核主要有四大内核，分别是Tirdent、Geckos、Presto、Webkit、 KHTML。Chrome浏览器（浏览器版本48.0.2564.109，内核版本WebKit 537.36）、Edge浏览器（浏览器版本 25.10586.0.0，内核版本Edge 13.10586）、IE浏览器（浏览器版本 11.545.10586.0，内核版本Trident/7.0）

6.2 项目的目录结构

项目的目录结构及目录功能如下：

（1）/doc

用于存放项目的说明文档

（2）/modules

用于存放项目的核心模块，包括文档页渲染模块，首页渲染模块，配置文件解析模块，文件读写模块。

（3）/post\_source

用于给用户放置自己要发布的文档以及文档中使用的图片，里面的md文件会被转译为HTML文件用于网络查看。

（4）/publish

这个文件夹就是最终用户访问的文件夹，生成的首页，文档页以及文档，图片，都会被放置在这个文件夹。

（5）/src

这个文件夹用于放置源码，例如Less文件或Typescript文件，在编译后它们会被传送到static文件夹中用作网页样式及网页脚本。

（6）/static

这个文件夹用于放置样式，脚本，图片，以及模板文件。

（7）app.js

该文件为项目的主流程文件。

（8）home-config.json

该文件为首页的配置文件，以JSON格式保存，home-render模块将会通过解析该文件来生成首页文件。

（9）site-config.js

该文件为网站的配置文件，用于设置网站的主题，名称等等。

（10）server.js

该文件为服务器文件，启动服务器使用户可以从网络来访问网站。

6.3 文档框架的的实现

6.3.1 文档框架模块概述

通过分析框架的功能和流程，我将框架的一些功能抽象为几个模块。parse.js负责解析配置文件，home-render.js负责对网站首页的渲染，catalog-render.js负责对网站目录进行解析渲染，write-file.js则负责对文件进行操作（读写及复制）。

6.3.2 具体实现

（1）配置解析模块

parse.js用来解析Json格式的配置文件，首先读取配置文件并解析JSON为JavaScript对象，并处理配置信息，最后将包含配置信息的对象返回。

具体代码如下：

// 本模块用于解析配置文件

const fs = require("fs");

// 写入配置文件

// 配置解析模块

exports.analysis\_config = function(config){

let out\_config = {};

// 路径信息

let css\_path = 'css/';

let code\_style\_path = 'css/code\_styles/';

// title解析

out\_config.title = config.title;

//theme解析

out\_config.theme = css\_path + config.theme + '.css';

// code\_style解析

out\_config.code\_theme = code\_style\_path + config.code\_theme + '.css';

//theme\_color解析

out\_config.theme\_color = config.theme\_color;

// catalog解析

out\_config.catalog = config.catalog;

// first\_show解析

out\_config.first\_show = config.first\_show;

return out\_config;

}

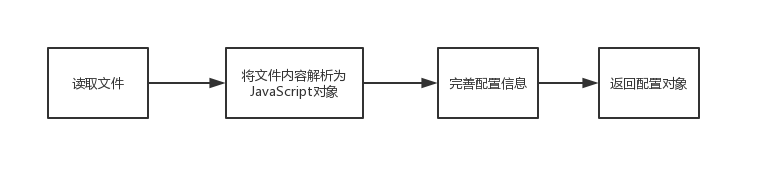


图3配置解析流程图

（2）渲染模块

home-render.js及catalog-render.js通过解析的配置文件，对网站首页及目录页进行渲染。以下是示例代码：

// 此模块用于解析和插入目录

const re = require('./re');

const fs = require('fs');

const cheerio = require('cheerio');

const pa = require("./parse");

const prime\_config = require('./../site-config.json');

const site\_config = pa.analysis\_config(prime\_config);

const home\_config = require('./../home-config.json');

// 加载模板

// 将模板加载为dom树

let $ = cheerio.load(fs.readFileSync('./static/template/guide.html'));

// 渲染目录

exports.renderCatalog = function(){

let catalog = site\_config.catalog;

// 渲染title及header

$('title').text(site\_config.title);

// header部分

function setHeader(){

setHeaderLogo();

setHeaderLinks();

}

// 对header中logo的解析

function setHeaderLogo(){

if(home\_config.header.logo){

let logo = home\_config.header.logo;

$('#header').append('<div class="logo header-left"></div>');

for(let i of logo){

if(i.type === 'img'){

$('#header .logo').append('<img>');

$('#header .logo img').attr('src',i.src);

}

if(i.type === 'icon'){

$('#header .logo').append('<i></i>');

$('#header .logo i').addClass('fa fa-' + i.icon\_name);

if(i.color){

$('#header .logo i').attr('style','color:' + i.color + ';');

}

}

if(i.type === 'text'){

$('#header .logo').append('<span></span>');

$('#header .logo span').text(i.content);

}

}

}

}

// 对header中links的解析

function setHeaderLinks(){

if(home\_config.header.links){

let links = home\_config.header.links;

$('#header').append('<div class="links"></div>');

// 对position进行解析

if(links.position){

if(links.position === 'left'){

$('#header .links').addClass('header-left');

}else if (links.position === 'right') {

$('#header .links').addClass('header-right');

}

}

// 对链接的解析

if(links.children){

for(let i of links.children){

// 对单链接的解析

if(i.type === 'link'){

let a = `<a href="${i.href}">${i.text}</a>`;

$('#header .links').append(a);

}

// 对下拉菜单的解析

if(i.type === 'dropdown'){

$('#header .links').append('<div class="dropdown"></div>');

let dropdown = `<a class="dropdown-nav">${i.text}</a>`;

$('#header .links .dropdown').append(dropdown);

$('#header .links .dropdown').append('<ul></ul>');

for(let children of i.children){

let a = `<li><a href="${children.href}">${children.text}</a></li>`;

$('#header .links .dropdown ul').append(a);

}

}

}

}

}

}

setHeader();

// 解析并生成目录

for(let i of catalog){

let title1 = cheerio.load('<div class="title1"></div>');

if(i.title && i.src){

let thisTitle = `<h3><span class="doc-link" data-src="${i.src}">${i.title}</span></h3>`;

title1('.title1').append(thisTitle);

}else{

let thisTitle = `<h3><span>${i.title}</span></h3>`;

title1('.title1').append(thisTitle);

}

if(i.children){

for(let j of i.children){

let title2 = cheerio.load('<div class="title2"></div>');

if(j.title && j.src){

let thisTitle = `<span class="doc-link" data-src="${j.src}">${j.title}</span>`;

title2('.title2').append(thisTitle);

}else{

let thisTitle = `<span>${j.title}</span>`;

title2('.title2').append(thisTitle);

}

if(j.children){

let title3 = cheerio.load('<div class="title3"></div>');

for(let k of j.children){

if(k.title && k.src){

let thisTitle = `<p class="doc-link" data-src="${k.src}">${k.title}</p>`;

title3('.title3').append(thisTitle);

}else{

let thisTitle = `<p>${k.title}</p>`;

title3('.title3').append(thisTitle);

}

}

title2('.title2').append(title3.html());

}

title1('.title1').append(title2.html());

}

}

$('#catalog').append(title1.html());

}

// 设置首个展示的文档

$('#container').append(`<iframe src="${site\_config.first\_show}" frameborder="0" id="doc">`)

let writeStream = $.html();

fs.writeFileSync("./publish/guide.html",writeStream);

}

（3）文件读写模块

write-file.js中主要是对文件操作的一些代码，如写入文件，复制文件等。以下是示例代码：

const fs = require("fs");

const json = require("./parse");

const re = require("./re");

const md = require("markdown-it")();

const less = require('less');

// 读取配置文件并解析配置项

const prime\_config = require('./../site-config.json');

const config = json.analysis\_config(prime\_config);

// 将配置项替换好写入一个md文件

exports.writeMd = function(file\_name){

const source\_path = "./post\_source/";

const publish\_path = "./publish/";

let out\_file\_name = file\_name.slice(0,-3) + ".html";

//首先读取模板文件

let template = fs.readFileSync("./static/template/md.html").toString();

// 读取md正文内容并解析为html

let prime\_content = fs.readFileSync(source\_path+file\_name).toString();

let content = md.render(prime\_content);

// 设置配置替换项

const md\_config = [

{

replace\_pattern:"theme",

value:config.theme

},{

replace\_pattern:"code\_theme",

value:config.code\_theme

},{

replace\_pattern:"content",

value:content

}

];

// 将模板文件中配置项替换掉

let writeStream = re.replaceAll(template,md\_config);

//写入文件

fs.writeFile(publish\_path + out\_file\_name,writeStream,function(err){

if(err){

console.log(out\_file\_name+'写入失败');

console.log(err);

}

else{

console.log(out\_file\_name+"写入成功");

}

})

}

// 将一个文件夹中的所有md文件转换为html

exports.writeAllMd = function(){

const source\_path = "./post\_source/";

//const publish\_path = "./publish/";

let files = fs.readdirSync(source\_path);

for(let i of files){

if(i.slice(-3) === '.md'){

exports.writeMd(i);

}

}

}

// 复制文件夹

exports.copyDir = function(src, dst){

console.log('复制' + src + '到' + 'dst' + '...');

if(!fs.existsSync(dst)){

fs.mkdirSync(dst);

}

fs.readdir(src, function(err, paths){

if(err){

throw err;

}

paths.forEach(function(path){

let \_src = src + '/' + path,

\_dst = dst + '/' + path,

st = fs.statSync(\_src);

if(st.isFile()){

fs.copyFileSync(\_src, \_dst);

}else if(st.isDirectory()){

if(!fs.existsSync(dst)){

fs.mkdirSync(\_dst);

}

exports.copyDir(\_src, \_dst);

}

})

})

}

// 复制文件(删除原来的文件)

exports.copyFile = function(src, dst){

try{

fs.copyFileSync(src, dst);

}catch(e){

fs.unlinkSync(dst);

}

}

// 将一个文件夹中的html文件原样复制

exports.copyHtml = function(src, dst){

let \_src = src + '/',

\_dst = dst + '/';

let files = fs.readdirSync(src);

for (let i of files){

if(i.slice(-5) === '.html'){

exports.copyFile(\_src + i, \_dst + i);

}

}

}

// 生成theme-color的函数

exports.generateThemeColor = function(file,color){

let template = `@theme-color: ${color};`;

fs.writeFileSync(file,template);

console.log('theme-color文件已生成...');

}

// 将less转换为css

exports.less = function(src, dst){

console.log(`编译${src}为css...`);

let lessFile = fs.readFileSync(src,'utf8');

less.render(lessFile,function(e,output){

fs.writeFileSync(dst,output.css);

console.log(`${src}编译完成...`);

})

}

6.4 服务器端的实现

6.4.1 Express框架概述

Express是一个高效而强大的基于Node.js平台，它提供一系列强大的特性，能够帮助开发者创建各种 Web 和移动设备应用。其次，Express是Node.js的 一个简洁、灵活的MVC幵发框架，它提供一系列强大的特性，具有丰富的HTTP 快捷方法和任意排列组合的Connect中间件[11]。让开发者创建健壮、友好的 API 变得既快速又简单。

6.4.2 具体实现

通过express的express.static 内置中间件函数来对文档提供静态访问。

以下是示例代码：

var express = require('express');

var app = express();

app.get('/', function (req, res) {

res.send('Hello World!');

});

app.use(express.static('publish'));

var server = app.listen(3000, function () {

var host = server.address().address;

var port = server.address().port;

console.log('Example app listening at http://%s:%s', host, port);

console.log(server.address());

});

7 程序测试

7.1 界面测试

7.1.1 浏览器界面兼容测试

以chrome，edge浏览器访问网站，兼容性良好，测试截图如下：

图 1 首页显示



图 12 文档页显示

8 结论

本项目经过两个月的间断开发，成功利用JavaScript，Express框架实现了需求分析中所列功能，并成功使用。在开发中存在许多问题，但也逐一解决了。经过部分使用用户的反馈，项目中程序的用户界面以及用户体验有待进一步优化和提升。后台开发在代码逻辑上也有待进一步的修改和完善，开发者也在积极筹备重构中。

参考文献

[1] GitHub Wikipedia [EB/OL]. (2019-3-20) [2018-3-25]. https://en.wikipedia.org/wiki/GitHub.

[2] W3school. HTML 基础教程[EB/OL]. (2012-5-1) [2018-3-25]. http://www.w3school.com.cn/html/index.asp.

[3] Eric A. Meyer. Cascading Style Sheets:The Definitive Guide [M]. United States of America: O’Reilly, 2018.

[4] Nicholas C. Professional JavaScript for Web Developers [M]. United States of America: Wiley E-Text, 2012.

[5] 陆凌牛. Node.js权威指南 [M]. 北京: 机械工业出版社, 2015.

[6] ShowDoc [EB/OL]. (2019-3-20) [2018-3-25]. <https://www.showdoc.cc/>

[7] 朴灵．深入浅出Node．JS[M]．北京：人民邮电出版社，2013：5—89．

[8] Markdown Wikipedia [EB/OL]. (2019-3-20) [2018-3-25]. https://en.wikipedia.org/wiki/Markdown

[9] 程桂花, 沈炜, 何松林等. Node.js中Express框架路由机制的研究[J]. 工业控制计算机, 2016, 08:101-102.

[10] 张博,于海洋. 服务器端JavaScript技术分析[J]. 信息与电脑(理论版), 2018, 04:19-20.

[11] 程桂花, 沈炜, 何松林等. Node.js中Express框架路由机制的研究[J]. 工业控制计算机, 2016, 08:101-102.

致谢

在本篇论文完成的同时，我首先要感谢我的导师潘勇浩老师。在潘勇浩老师的悉心指导下，我的论文才能顺利的完成。在本篇论文的完成过程中，潘勇浩老师对我提出了很多建设性的指导意见，细心地和我讲解论文中出现的问题，给予专业上的指导，信心上的鼓励。我能顺利完成本篇论文和老师的认真指导和辛苦的批改是分不开的，在此我由衷的向潘勇浩老师表示感谢。

我也要感谢我的室友，他们在我写作的这段时间提供了良好的写作环境和各种有利条件，不断鼓励和帮助我，让我专注于写作一事而不用关注琐事。

最后，还要感谢参加评阅论文和答辩评审的各位专家老师，感谢你们在忙碌的日程中抽空参与我们的毕业事宜。