<https://segmentfault.com/a/1190000006939687>

### 项目来源

以前曾用过WordPress搭建自己的博客网站，但感觉WordPress很是臃肿。所以一直想自己写一个博客内容管理器。

正好近日看完了Vue各个插件的文档，就用着Vue尝试写了这个简约的博客内容管理器（CMS）。

### 嗯，我想完成的功能：

* 一个基本的博客内容管理器功能，如后台登陆，发布并管理文章等
* 支持markdown语法实时编辑
* 支持代码高亮
* 管理博客页面的链接
* 博客页面对移动端适配优化
* 账户管理(修改密码)

#### [Demo](http://115.28.90.175/)

登陆后台按钮在页面最下方“站长登陆”，可以以游客身份登入后台系统。

#### [源码](https://github.com/ycwalker/CMS-of-Blog)

### 用到的技术和实现思路：

#### 前端：Vue全家桶

* Vue.js
* Vue-Cli
* Vue-Resource
* Vue-Router
* Vuex

#### 后端：Node

* Node.js
* mongoDB (mongoose)
* Express

#### 工具和语言

* Webpack
* ES6
* SASS

#### 整体思路：

* Node服务端不做路由切换，这部分交给Vue-Router完成
* Node服务端只用来接收请求，查询数据库并用来返回值

所以这样做前后端几乎完全解耦，只要约定好restful数据接口，和数据存取格式就OK啦。

后端我用了mongoDB做数据库，并在Express中通过mongoose操作mongoDB，省去了复杂的命令行，通过Javascript操作无疑方便了很多。

#### Vue的各个插件：

* vue-cli:官方的脚手架，用来初始化项目
* vue-resource：可以看作一个Ajax库，通过在跟组件引入，可以方便的注入子组件。子组件以this.$http调用
* vue-router：官方的路由工具，用来切换子组件，是用来做SPA应用的关键
* vuex：规范组件中数据流动，主要用于异步的http请求后数据的刷新。通过官方的vue-devtools可以无缝对接

### 文件目录

│ .babelrc babel配置

│ .editorconfig

│ .eslintignore

│ .eslintrc.js eslintrc配置

│ .gitignore

│ index.html 入口页面

│ package.json

│ README.md

│ setup.html 初始化账户页面

│ webpack.config.js webpack配置

│

├─dist 打包生成

│

├─server 服务端

│ api.js Restful接口

│ db.js 数据库

│ index.js

│ init.json 初始数据

│

└─src

│ main.js 项目入口

│ setup.js 初始化账户

│

├─assets 外部引用文件

│ ├─css

│ ├─fonts

│ ├─**img**

│ └─js

│

├─components vue组件

│ ├─back 博客控制台组件

│ ├─front 博客页面组件

│ └─share 公共组件

│

├─router 路由

│

├─store vuex文件

│

└─style 全局样式

前端的文件统一放到了src目录下，有两个入口文件，分别是main.js和setup.js，有过WordPress经验应该知道，第一次进入博客是需要设置用户名密码和数据库的，这里的setup.js就是第一次登入时的页面脚本，而main.js则是剩余所有文件的入口

#### main.js

**import** Vue **from** 'vue'

**import** VueResource **from** 'vue-resource'

**import** {mapState} **from** 'vuex'

//三个顶级组件，博客主页和控制台共享

**import** Spinner **from** './components/share/Spinner.vue'

**import** Toast **from** './components/share/Toast.vue'

**import** MyCanvas **from** './components/share/MyCanvas.vue'

**import** store **from** './store'

**import** router **from** './router'

**import** './style/index.scss'

Vue.use(VueResource)

**new** Vue({

router,

store,

components: {Spinner, Toast, MyCanvas},

computed: mapState(['isLoading', 'isToasting'])

}).$mount('#CMS2')

而后所有页面分割成一个单一的vue组件，放在components中，通过入口文件main.js，由webpack打包生成，生成的文件放在dist文件夹下。

后端文件放在server文件夹内，这就是基于Express的node服务器，在server文件夹内执行

**node** **index**

就可以启动Node服务器，默认侦听3000端口。

### 关于 Webpack

Webpack的配置文件主体是有vue-cli生成的，但为了配合后端自动刷新、支持Sass和生成独立的css文件，稍微修改了一下：

#### webpack.config.js

**const** path = require('path')

**const** webpack = require('webpack')

**const** ExtractTextPlugin = require('extract-text-webpack-plugin')

**const** CopyWebpackPlugin = require('copy-webpack-plugin')

//萃取css文件，在此命名

**const** extractCSSFromVue = **new** ExtractTextPlugin('styles.css')

**const** extractCSSFromSASS = **new** ExtractTextPlugin('index.css')

module.exports = {

entry: {

main: './src/main.js',

setup: './src/setup.js'

},

output: {

path: path.resolve(\_\_dirname, './dist'),

publicPath: '/dist/',

filename: '[name].js'

},

resolveLoader: {

moduleExtensions: ['-loader']

},

module: {

rules: [

{

test: /\.vue$/,

loader: 'vue',

//使用postcss处理加工后的scss文件

options: {

preserveWhitespace: false,

postcss: [

require('autoprefixer')({

browsers: ['last 3 versions']

})

],

loaders: {

sass: extractCSSFromVue.extract({

loader: 'css!sass!',

fallbackLoader: 'vue-style-loader'

})

}

}

},

{

test: /\.scss$/,

loader: extractCSSFromSASS.extract(['css', 'sass'])

},

{

test: /\.js$/,

loader: 'babel',

exclude: /node\_modules/

},

{

test: /\.(png|jpg|gif|svg)$/,

loader: 'file',

options: {

name: '[name].[ext]?[hash]'

}

},

//字体文件

{

test: /\.woff(2)?(\?v=[0-9]\.[0-9]\.[0-9])?$/,

loader: 'url-loader?limit=10000&mimetype=application/font-woff'

},

{

test: /\.(ttf|eot|svg)(\?v=[0-9]\.[0-9]\.[0-9])?$/,

loader: 'file-loader'

}

]

},

plugins: [

//取出css生成独立文件

extractCSSFromVue,

extractCSSFromSASS,

**new** CopyWebpackPlugin([

{from: './src/assets/img', to: './'}

])

],

resolve: {

alias: {

'vue$': 'vue/dist/vue'

}

},

//服务器代理，便于开发时所有http请求转到node的3000端口，而不是前端的8080端口

devServer: {

historyApiFallback: true,

noInfo: true,

proxy: {

'/': {

target: 'http://localhost:3000/'

}

}

},

devtool: '#eval-source-map'

}

**if** (process.env.NODE\_ENV === 'production') {

module.exports.devtool = '#source-map'

module.exports.plugins = (module.exports.plugins || []).concat([

**new** webpack.DefinePlugin({

'process.env': {

NODE\_ENV: '"production"'

}

}),

**new** webpack.optimize.UglifyJsPlugin({

compress: {

warnings: false

}

}),

**new** webpack.LoaderOptionsPlugin({

minimize: true

})

])

}

运行

npm start

后，node端开启了3000端口，接着运行

npm **run** dev

打开webpack在8080端口服务器，具有动态加载的功能，并且所有的http请求会代理到3000端口

### 关于Vue-Router

因为写的是但也应用（SPA），服务器不负责路由，所以路由方面交给Vue-Router来控制。

#### router.js

**import** Vue **from** 'vue'

**import** Router **from** 'vue-router'

//博客页面

**import** Archive **from** '../components/front/Archive.vue'

**import** Article **from** '../components/front/Article.vue'

//控制台页面

**import** Console **from** '../components/back/Console.vue'

**import** Login **from** '../components/back/Login.vue'

**import** Articles **from** '../components/back/Articles.vue'

**import** Editor **from** '../components/back/Editor.vue'

**import** Links **from** '../components/back/Links.vue'

**import** Account **from** '../components/back/Account.vue'

Vue.use(Router)

**export** **default** **new** Router({

mode: 'history',

routes: [

{path: '/archive', name: 'archive', component: Archive},

{path: '/article', name: 'article', component: Article},

{path: '/', component: Login},

{

path: '/console',

component: Console,

children: [

{path: '', component: Articles},

{path: 'articles', name: 'articles', component: Articles},

{path: 'editor', name: 'editor', component: Editor},

{path: 'links', name: 'links', component: Links},

{path: 'account', name: 'account', component: Account}

]

}

]

})

### 文档首页

#### index.html

**<!DOCTYPE html>**

<html lang="en">

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>cms2simple</title>

<link rel="stylesheet" href="dist/index.css">

<link rel="stylesheet" href="dist/styles.css">

</head>

<body>

<div id="CMS2" style="height: 100%">

<my-canvas></my-canvas>

<spinner v-show="isLoading"></spinner>

<Toast v-show="isToasting"></Toast>

<router-view ></router-view>

</div>

<script src="/dist/main.js"></script>

</body>

</html>

可以看到路由控制在body元素下的router-view中。前面的spinner，toast元素分别是等待效果（转圈圈）的弹出层和信息的弹出层，和背景样式的切换。

### 关于后端

后端是用node.js作为服务器的，使用了express框架。

其中代码非常简单:

#### index.js

**const** fs = require('fs')

**const** path = require('path')

**const** express = require('express')

**const** favicon = require('serve-favicon')

**const** bodyParser = require('body-parser')

**const** cookieParser = require('cookie-parser')

**const** db = require('./db')

**const** resolve = file => path.resolve(\_\_dirname, file)

**const** api = require('./api')

**const** app = express()

// const createBundleRenderer = require('vue-server-renderer').createBundleRenderer

app.set('port', (process.env.port || 3000))

app.use(favicon(resolve('../dist/favicon.ico')))

app.use(bodyParser.json())

app.use(bodyParser.urlencoded({extended: false}))

app.use(cookieParser())

app.use('/dist', express.static(resolve('../dist')))

app.use(api)

app.post('/api/setup', **function** (req, res) {

**new** db.User(req.body)

.save()

.then(() => {

res.status(200).end()

db.initialized = true

})

.catch(() => res.status(500).end())

})

app.get('\*', **function** (req, res) {

**const** fileName = db.initialized ? 'index.html' : 'setup.html'

**const** html = fs.readFileSync(resolve('../' + fileName), 'utf-8')

res.send(html)

})

app.listen(app.get('port'), **function** () {

console.log('Visit http://localhost:' + app.get('port'))

})

服务器做的事情很简单，毕竟路由在前端。在接受请求的时候判断一下数据库是否初始化，如果初始化就转向主页，否则转向setup.html，之所以没有直接sendfile是因为考虑到之后添加服务端渲染（虽然主页并没有啥值得渲染的，因为很简单）

express框架中使用了mongoose来连接mongoDB数据库，在接收请求时做对应的curd操作，比如这就是在接收保存文章时对应的操作：

#### api.js

router.post('/api/saveArticle', (req, res) => {

**const** id = req.body.\_id

**const** article = {

title: req.body.title,

date: req.body.date,

content: req.body.content

}

**if** (id) {

db.Article.findByIdAndUpdate(id, article, fn)

} **else** {

**new** db.Article(article).save()

}

res.status(200).end()

})

### 后记

当然还有很多没提及的地方，最早写这个博客管理器的时候用的还是vue 1.x，后来用2.0改写后文档一直没改，所以最近更新了一下，避免误解。

其实整个管理器最复杂的地方时vuex异步数据视图的部分，不过这一部能讲的太多，就不在这里展开了，可以看官方文档后，参考源代码的注释。