文章编号:1007-1423(2017)30-0051-05 DOI:10.3969/j.issn.1007-1423.2017.30.011

基于 Vue.js 的后台单页应用管理系统的研究与实现

旷志光,纪婷婷,吴小丽

(上海计算机软件技术开发中心,上海 201112)

摘要:

通过研究 Vue.js、路由、全局状态、axios 等技术,实现后台单页应用管理系统。对 actions 进行封装,并在配置文件中定义好相关请求信息,即可实现带后台 API 请求的 action 操作,极大程度减少系统代码量,并保证系统的统一性、健壮性。前端页面根据菜单策略控制页面文件的入口,根据页面权限控制策略进行按钮级别权限校验。

关键词:

Vue.js; MVVM; 单页应用; 权限管理

基金项目:

上海市软件技术创新服务平台(No.17DZ2292100)

0 引言

为了满足日益复杂、多样的 Web App 需求,越来越多的原本后端处理的业务逻辑开始转移到前端来处理。Vue.JS 就是这样一套构建用户界面的渐进式框架,方便大部分后端逻辑移植到前端去实现。Vue.JS 采用自底向上增量开发的设计,其核心库只关注视图层,它不仅易于上手,还便于与第三方库或既有项目整合。另一方面,当 Vue.JS 与单文件组件及其 Vue.JS 生态系统支持的库结合使用时,也完全能够为复杂的单页应用程序提供驱动。由此,采用 Vue.JS 进行后台管理系统的前端设计工作,一定程度减轻前后端开发人员开发难度。

1 系统关键组件选型

本系统前端工程基于流程简化、低成本、快速开发、高性能等需求,主要选用到 Vue.JS、vue-router、vuex、axios 等关键组件。

Vue.JS 是整个前端工程的基础视图层框架,主要解决前端数据绑定的问题。传统的前端开发,主要基于 jQuery 通过各种复杂的选择器来操作 DOM。同时,通过 AJAX 跟服务器请求数据,前端代码一层层解析 JSON,将 JSON 某个层级的数据赋给相应的 DOM 操作,还要进行请求的异常处理,数据不但操作繁琐复杂且易出现未知错误。而通过 Vue.JS 的响应式双向绑定数据,实时反映数据的真实变化并映射到目标虚拟 DOM 上,避免前端页面开发中 DOM 选择器繁杂的操作,简化 Web 前端开发流程和降低开放难度,提升前端开发效率,降低开发成本和周期。

vue-router 是 Vue.JS 官方发布的一款路由插件,和 Vue.JS 是深度集成的,适合用于构建单页面应用。传统的页面应用,是用一些超链接来实现页面切换和跳转的。而在 vue-router 单页面应用中,则是路径之间的切换,也就是组件的切换。通过 router-view 来动态挂载页面组件,并最终渲染成页面。另外,HTML5 里引入了新的 API,history.pushState 和 history.replaceState,可以通过这个新的接口做到无刷新访问页面的同时改变页面 URL[1],这让 Vue.JS 能够动态调整页面路径,方便页面切换,提高了用户体验。

vuex 是 Vue.JS 官方依照 Flux 实现的一套全局状态管理方案,并被集成到 vue-devtools,无需配置即可在浏览器中实现时光旅行式调试。当开发单页应用时,我们通常会把状态储存在组件的内部,每一个组件

都拥有其自身的状态管理，但是在整个应用层面上看，很多公共的状态是却是分散在各个页面中。同时，我们经常会需要把状态的一部分共享给多个组件。一个常见的解决方案是使用事件系统来让一个组件把一些状态“告知”到其他组件中，来让相应的组件去响应变化。但是这种模式的问题在于，大型组件树中的事件流会很快变得非常繁杂，并且调用时很难去找出究竟哪里出错，事件的冒泡也很有可能导致整个应用的资源消耗非常大。应用越来越大，页面文件数也越来越多，多个状态分散的跨越在许多组件和交互中，其复杂度也经常逐渐增长。使用 vuex 将状态放入一个全局的实例中，做到各个组件同步响应，减少系统状态复杂度。但是在使用全局状态管理时，我们还需要对组件的组件本地状态（component local state）和应用层级状态（application level state）进行区分，避免出现组件本地状态放到应用级状态去管理。应用级的状态不属于任何特定的组件，但每一个组件仍然可以监视（Observe）其变化从而响应式地更新 DOM。

通过 vue-router 我们解决了页面切换问题，通过 vuex 我们解决了全局状态共享管理的问题，但是所有应用的基础在于数据。传统数据请求，主要利用 jQuery 封装的 AJAX 请求来实现。在处理异步问题时，一般采用的是 callback 回调的方式。callback 回调存在一个很严重的金字塔问题——大量的回调函数慢慢向右侧屏幕延伸的一种状态[2]。通过采用含 Promise 特性的组件来进 AJAX 请求，使得我们可以用同步的代码形式实现异步的请求操作。axios 就是这样一个基于 Promise 用于浏览器和 nodejs 的 HTTP 客户端，其可以从浏览器中创建 XMLHttpRequest，支持 Promise API，同时方便实现请求过程中的中间件操作，例如权限校验。

整个前端工程总结下来，主要采用 vue 作为基础视图层框架，利用 vue-router 完成前端页面路由的跳转及各类访问拦截功能，采用 axios 作为 HTTP 请求库，同时利用 vuex 负责前端的全局状态管理，采用 iView 作为界面基础组件库。

2 架构设计

2.1 前端结构

整个后台单页应用管理系统分为两个工程：前端工程、后端工程，以 API 接口形式联合前后端。其中前端工程的结构目录如图 1 所示：

```
src
├── App.vue
├── api
│   ├── config.js          # action与api请求配置
│   ├── helper.js          # ajax请求的封装方法
│   └── status.js          # 通用状态提示码
├── common
│   ├── config.js          # 全局配置项
│   ├── mutation-types.js  # mutation-types定义
│   └── permission.js      # 权限定义
├── components
│   └── ...                # 组件目录
├── main.js                # 入口文件
├── pages
│   └── ...                # 页面目录
├── router
│   ├── config.js          # 路由配置
│   └── index.js
└── store
    ├── actions.js         # 遍历生成带ajax请求的action
    ├── actions_ext.js     # 不带api请求的action
    ├── getters            # getters
    ├── index.js           # vuex入口文件
    └── modules            # mutations
```

图 1 前端工程结构

2.2 详细设计

前端工程中，以 components 作为独立页面组件的存放目录，使用 pages 作为系统页面的存放目录，所有涉及全局状态管理的代码存放在 store 目录，涉及路由配置的代码存放在 router 目录下，通用的配置文件存放在 common 目录。

通常，在进行 store 设计时，会将所有的 action 集中在 action.js 文件中，其中包含通过 axios 进行后台 api 请求的代码。在进行后台单页应用系统开发的时候，会频繁跟后台进行数据沟通，故考虑将 action 分成两种类型，一种是不带后台 API 请求的，只是单纯进行前端操作的，另外一直则是包含后台 API 请求，跟后台进行数据沟通。根据以往经验，大概 2/3 的 action 会涉及后台 API 请求，故需要通过某种形式来减少重复的请求代码。经过对上述问题的思考，可以通过以下方式来减少系统的代码量，同时保证请求统一性，尤其是请求异常的统一性处理：

在 common/mutation-types.js 中声明所有的 action 的 name，如：export const BACKEND_LOGIN = 'BACKEND_LOGIN'，其中定义了一个 BACKEND_LOGIN 的 action。

在 api/config.js 中声明所有涉及后台 API 请求的 action，并配置其请求地址等内容，其格式如下：api [BACKEND_LOGIN] = { url: ´some_url´, method: ´get´, pathinfo: truelfalse, noMutation: truelfalse }。其中，对象的键是 mutation-type 的常量，url 表示请求的路径，method 表示请求的方式，pathinfo 表示请求的 url 参数是否以 pathinfo 加入到请求的路径上，noMutation 表示是否忽略 commit 操作，该设计考虑是，某些 ajax 请求到的数据，并非全局的状态，而是某些页面级别的状态，故无需将请求到的相关数据进行全局状态管理。

在 api/helper.js 中，声明 createAPIRequest 方法，其接受三个参数：commit，type，param，其中 commit 是一个方法，进行 mutation 的 commit 操作，type 是一个具体的 mutation-type（即 action 的 name），param 是需要传入的请求参数。通过传入的 type，可以从 api/config.js 中获取该 type 的 api 请求配置。根据其配置，进行相应的后台 api 请求，并根据配置项中的 noMutation 来鉴别是否需要对 mutation 进行 commit 操作，最终返回一个 Promise 实例，使得我们可以使用同步代码的形式实现各类异步操作。核心代码如下所示：

```
// api/config.js
import api from ´./config´
export const createAPIRequest = function (commit, type, params = {}) {
    let apiItem = api[type]
    if (apiItem) {
        // request config
        let config = {}
        config.url = API_ROOT + apiItem.url
        config.method = apiItem.method || ´get´
        config.headers = {}
        // 以下省略处理 params，根据请求 method 决定
        ......
        // do request
        return axios(config)
        .then(function (response) {
            return response.data
        }, function ({response}) {
            // 以下省略网络异常处理
            ......
        })
        .then(function (data) {
        // 根据 noMutation 决定是否要进行 commit
        if (apiItem.noMutation === undefined || apiItem.noMutation === false) {
            commit(type, data)
        }
        return data
    })
    } else {
        // none api defined
        return new Promise(function (resolve, reject) {
            reject(ERROR_NO_API_CONFIG)
        })
    }
}
```

最终，我们在 store/actions.js 中引入上述 3 个文件，并通过遍历 config，动态生成带后台 api 请求的 action。相关代码参考如下：

```
import {createAPIRequest} from ´../api/helper´
import config from ´../api/config´
// create api actions,use mutions type as action name
let actions = {}
Object.keys(config).forEach(function (type) {
    actions[type] = function ({commit}, params = {}) {
        return createAPIRequest(commit, type, params)
    }
})
```

其中 actions 包含所有带 API 请求的 action，通过跟不带 API 请求的 action 进行合并，得到这个 store 管理需要使用的 action。

2.3 权限设计

另外，在进行前端系统设计的时候，还需考虑整个系统的权限问题。在后台应用中，主要是在中间件中根据 RBAC 来鉴别用户权限[3]。在前端页面中，主要分以下三个层面进行用户权限的控制：

第一层，主要是对用户入口的控制，表现为：根据用户角色来获取用户所能访问的菜单列表。

第二层，主要是对用户访问页面的时候，进行权限鉴别。在 router 的全局钩子中，根据用户角色的权限，来鉴别用户是否拥有访问该页面的权限，如果有权限，则 next()，否则，直接重定向用户到 401 页面。

第三层，当用户只有访问页面权限，但没有进行操作权限时，通过封装的 hasPermission 方法动态展示或隐藏相关操作入口，也可以根据实际请求展示或隐藏相关数据列。

3 部署实现

通过前后端分离，来保证前后端发布的独立性。整个应用的请求反馈逻辑如图 2 所示：

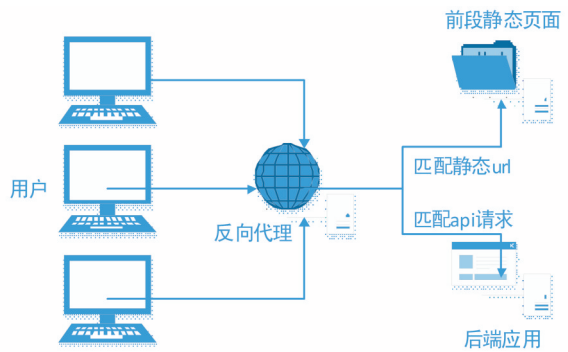

图2 部署结构

针对前端工程，利用 webpack 打包来生成生成环境下的前端代码。针对后端工程，则需根据特定的开发环境，打包相应的生产环境应用。当进行前后端分离发布，也经常会遇到跨域的问题，为减少代码层面的修改动作，可利用 Nginx 进行反向代理，将两个工程的放到一个域名下去部署，从而解决跨域问题。以下是以 PHP 应用为样例的 Nginx 配置文件，其主要思想是将对前端页面的请求反向代理到前端静态页面，由前端 vue-router 进行页面路由控制；将所有对 QPI 的访问反向代理到后台 PHP 应用。关键配置如下：

```
server {
    listen 91;
    set $root /usr/share/nginx/ admin/public;
    location ~ .*\.(gif|jpg|jpeg|bmp|png|ico|txt|js|css)$
    {
        root $root;
    }
    location / {
        root $root;
        index index.html;
        if ( -f $request_filename) {
            break;
        }
        if ( !-e $request_filename) {
            rewrite ^(.*)$ /index.php/$1 last;
            break;
        }
    }
    location /api {
        root $root;
        index  index.html index.php;
        if ( -f $request_filename) {
            break;
        }
        if ( !-e $request_filename) {
            rewrite ^/api/(.*)$ /api/index.php/$1 last;
            break;
        }
    }
    ### vuejs 前端 URL 前缀
    location /system {
        index index.html;
        try_files $uri $uri/ @backendApp;
    }
    ### 将所有对 vuejs URL 的访问重定向到 index.html 文件
    location @backendApp {
        rewrite ^.*$ /index.html last;
    }
    location ~ .+\.php($|/) {
        include  fastcgi_params;
        fastcgi_pass unix:/run/php-fpm/php7.0-fpm.sock;
        fastcgi_split_path_info ^((?U).+.php)(/?.+)$;
        fastcgi_param PATH_INFO $fastcgi_path_info;
        fastcgi_param PATH_TRANSLATED $root$fastcgi_path_info;
        fastcgi_param SCRIPT_FILENAME $root$fastcgi_script_name;
    }
}
```

4 结语

本文主要从组件的选型、系统结构设计以及部署实现三个方面，详细描述基于 Vue.JS 进行后台单页应

用系统的设计工作。对相关组件的封装，并以配置的形式主导功能的开发，在一定程度上减少后台系统的前端开发工作量。利用 Vue.JS 的双向绑定特性，高效地将数据反映到页面模型上，同时 Vue.JS 更高效地处理页面 DOM 操作，提升后台应用的性能。

参考文献：

[1]岳晓瑞, 陈继华. HTML5 环境下 PJAX 快速浏览技术实践[J]. 广东通信技术, 2013.

[2]邓森泉, 杨海波. Promise 方式实现 Node.js 应用的实践[J]. 计算机系统应用, 2017.

[3]乔颖, 须德. 一种基于角色访问控制(RBAC)的新模型及其实现机制[J]. 计算机研究与发展, 2000.

作者简介：

旷志光(1991-)，男，江西吉安人，初级工程师，本科，研究方向为软件工程、云计算、数据挖掘

纪婷婷(1989-)，女，江苏徐州人，中级工程师，硕士研究生，研究方向为大数据、数据挖掘、可视化

吴小丽(1988-)，女，江苏南通人，初级工程师，本科，研究方向为数据库、软件工程

收稿日期：2017-08-22　　修稿日期：2017-10-15

Research and Practice of Single Page Application For Management System Based on Vue.js

KUANG Zhi-guang, JI Ting-ting, WU Xiao-li

(Shanghai Development Center of Computer Software Technology, Shanghai 201112)

Abstract：

By studying Vue.js, router, global state, axios and other components, achieves a single-page application for management system. Realizes the actions which contain API request, just encapsulates the actions and defines the relevant request information in the configuration file. Also, it will minimize the amount of system code, ensure the unity and robustness of the system. The front page controls the entry of the page files according to the menu policy, and the button level permission check according to the page permission control policy.

Keywords：

Vue.js; MVVM; Single Page Application; RBAC