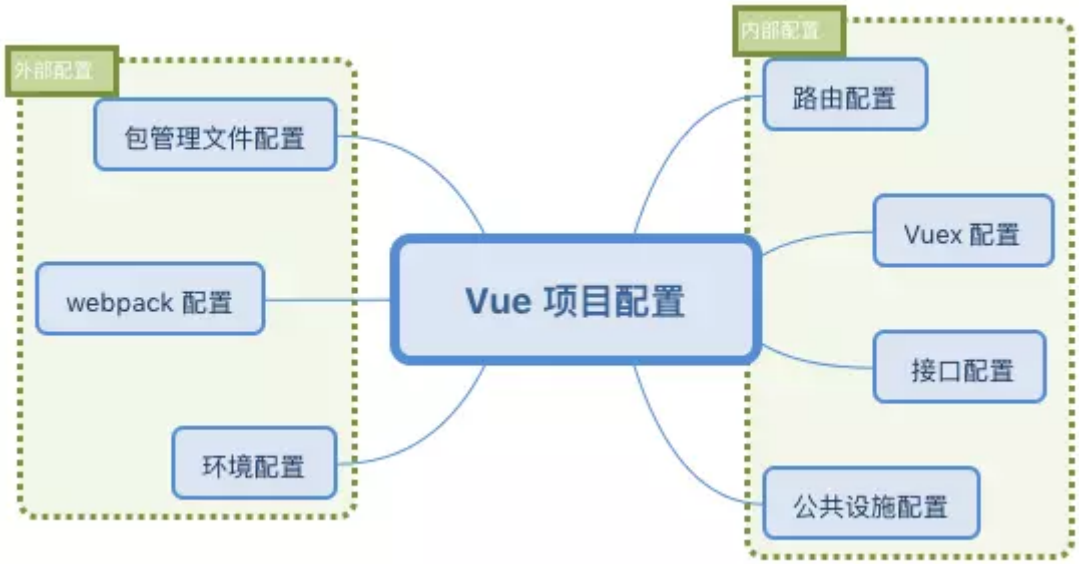
构建实战篇1：单页应用的基本配置

前几篇文章我们介绍了Vue 项目构建及运行的前期工作，包括webpack 的配置、环境变量的使用等，在了解并掌握了这些前期准备工作后，那么接下来我们可以走进Vue 项目的内部，一探其内部配置的基本构成。



配置

1. 路由配置

由于Vue 这类型的框架都是以一个或多个单页构成，在单页内部跳转并不会重新渲染HTML 文件，其路由可以由前端进行控制，因此我们需要在项目内部编写相应的路由文件，Vue 会解析这些文件中的配置并进行对应的跳转渲染。

我们来看一下CLI 给我们生成的router.js 文件的配置：

javascript

```
/* router.js */
```

```
import About from './views/About.vue' // 引入 About 组件

Vue.use(Router) // 注册路由

export default new Router({
    routes: [{
        path: '/',
        name: 'home',
        component: Home
    }, {
        path: '/about',
        name: 'about',
        component: About
    }]
})
```

这份配置可以算是最基础的路由配置，有以下几点需要进行优化：

- 如果路由存在二级目录，需要添加base 属性，否则默认为"/"
- 默认路由模式是hash 模式，会携带# 标记，与真实url 不符，可以改为history 模式
- 页面组件没有进行按需加载，可以使用 `require.ensure()` 来进行优化

下面是我们优化结束的代码：

javascript

```
/* router.js */

import Vue from 'vue'
import Router from 'vue-router'

// 引入 Home 组件
const Home = resolve => {
    require.ensure(['./views/Home.vue'], () => {
        resolve(require('./views/Home.vue'))
    })
}

// 引入 About 组件
const About = resolve => {
    require.ensure(['./views/About.vue'], () => {
        resolve(require('./views/About.vue'))
```

```
Vue.use(Router)

let base = `${process.env.BASE_URL}` // 动态获取二级目录

export default new Router({
    mode: 'history',
    base: base,
    routes: [{
        path: '/',
        name: 'home',
        component: Home
    }, {
        path: '/about',
        name: 'about',
        component: About
    }]
})
```

改为 history 后我们 url 的路径就变成了 `http://127.0.0.1:8080/vue/about`，而不是原来的 `http://127.0.0.1:8080/vue/#/about`，但是需要注意页面渲染 404 的问题，具体可查阅：HTML5 History 模式。

而在异步加载的优化上，我们使用了 webpack 提供的 require.ensure() 进行了代码拆分，主要区别在于没有优化前，访问 Home 页面会一起加载 About 组件的资源，因为它们打包进了一个 app.js 中：

但是优化过后，它们分别被拆分成了 2.js 和 3.js：

如此，只有当用户点击了某页面，才会加载对应页面的 js 文件，实现了按需加载的功能。

> webpack 在编译时，会静态地解析代码中的 require.ensure()，同时将模块添加到一个分开的 chunk 当中。这个新的 chunk 会被 webpack 通过 jsonp 来按需加载。

关于 `require.ensure()` 的知识点可以参考官方文档：require.ensure。

当然，除了使用 require.ensure 来拆分代码，Vue Router 官方文档还推荐使用动态 `import` 语法来进行代码分块，比如上述 require.ensure 代码可以修改为：

```javascript
// 引入 Home 组件
const Home = () => import('./views/Home.vue');

// 引入 About 组件
const About = () => import('./views/About.vue');
```

其余代码可以保持不变，仍然可以实现同样的功能。如果你想给拆分出的文件命名，可以尝试一下 webpack 提供的 `Magic Comments` （魔法注释）：

```javascript
const Home = () => import(/* webpackChunkName:'home'*/ './views/Home.vue');
```

2. Vuex 配置

除了 vue-router，如果你的项目需要用到 Vuex ，那么你应该对它有一定的了解，Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。这里我们先来看一下使用 CLI 生成的配置文件 store.js 中的内容：

```javascript
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex)
```

```
    },
    mutations: {

    },
    actions: {

    }
})
```

该配置文件便是 Vuex 的配置文件，主要有 4 个核心点：state、mutations、actions 及 getter，详细的介绍大家可以参考官方文档：核心概念，这里我用一句话介绍它们之间的关系就是：**我们可以通过 actions 异步提交 mutations 去 修改 state 的值并通过 getter 获取。**

需要注意的是不是每一个项目都适合使用 Vuex，如果你的项目是中大型项目，那么使用 Vuex 来管理错综复杂的状态数据是很有帮助的，而为了后期的拓展性和可维护性，这里不建议使用 CLI 生成的一份配置文件来管理所有的状态操作，我们可以把它拆分为以下目录：

```
└── store
    ├── index.js          # 我们组装模块并导出 store 的地方
    ├── actions.js        # 根级别的 action
    ├── mutations.js      # 根级别的 mutation
    └── modules
        ├── moduleA.js    # A模块
        └── moduleB.js    # B模块
```

与单个 store.js 文件不同的是，我们按模块进行了划分，每个模块中都可以包含自己 4 个核心功能。比如模块 A 中：

```
const moduleA = {
    state: {
        text: 'hello'
    },
    mutations: {
        addText (state, txt) {
            // 这里的 `state` 对象是模块的局部状态
            state.text += txt
        }
    },

    actions: {
        setText ({ commit }) {
            commit('addText', ' world')
        }
    },

    getters: {
        getText (state) {
            return state.text + '!'
        }
    }
}

export default moduleA
```

上方我们导出 A 模块，并在 index.js 中引入：

javascript

```
/* index.js */

import Vue from 'vue'
import Vuex from 'vuex'
import moduleA from './modules/moduleA'
import moduleB from './modules/moduleB'
import { mutations } from './mutations'
import actions from './actions'

Vue.use(Vuex)

export default new Vuex.Store({
    state: {
        groups: [1]
```

```
        moduleB, // 引入 B 模块
    },
    actions, // 根级别的 action
    mutations, // 根级别的 mutations

    // 根级别的 getters
    getters: {
        getGroups (state) {
            return state.groups
        }
    }
})
```

这样项目中状态的模块划分就更加清晰，对应模块的状态我们只需要修改相应模块文件即可。详细的案例代码可参考文末 github 地址。

3. 接口配置

在项目的开发过程中，我们也少不了与后台服务器进行数据的获取和交互，这一般都是通过接口完成的，那么我们如何进行合理的接口配置呢？我们可以在 src 目录下新建 services 文件夹用于存放接口文件：

```
└── src
    └── services
        ├── http.js      # 接口封装
        ├── moduleA.js    # A模块接口
        └── moduleB.js    # B模块接口
```

为了让接口便于管理，我们同样使用不同的文件来配置不同模块的接口，同时由于接口的调用 ajax 请求代码重复部分较多，我们可以对其进行简单的封装，比如在 http.js 中（fetch为例）：

javascript

```
/* http.js */
import 'whatwg-fetch'

// HTTP 工具类
export default class Http {
```

```
            headers: {
                'Content-Type': 'application/json'
            }
        };

        if (method === 'GET') {
            url += this.formatQuery(data)
        } else {
            param['body'] = JSON.stringify(data)
        }

        // Tips.loading(); // 可调用 loading 组件

        return fetch(url, param).then(response => this.isSuccess(response))
                .then(response => {
                    return response.json()
            })
    }

    // 判断请求是否成功
    static isSuccess(res) {
        if (res.status >= 200 && res.status < 300) {
            return res
        } else {
            this.requestException(res)
        }
    }

    // 处理异常
    static requestException(res) {
        const error = new Error(res.statusText)

        error.response = res

        throw error
    }

    // url处理
    static formatQuery(query) {
        let params = [];

        if (query) {
            for (let item in query) {
```

```
                }
            }
        }
        return params.length ? '?' + params.join('&') : '';
    }

    // 处理 get 请求
    static get(url, data) {
        return this.request('GET', url, data)
    }

    // 处理 put 请求
    static put(url, data) {
        return this.request('PUT', url, data)
    }

    // 处理 post 请求
    static post(url, data) {
        return this.request('POST', url, data)
    }

    // 处理 patch 请求
    static patch(url, data) {
        return this.request('PATCH', url, data)
    }

    // 处理 delete 请求
    static delete(url, data) {
        return this.request('DELETE', url, data)
    }
}
```

封装完毕后我们在 moduleA.js 中配置一个 github 的开放接口：

https://api.github.com/repos/octokit/octokit.rb

javascript

```
/* moduleA.js */
import Http from './http'

// 获取测试数据
export const getTestData = () => {
```

然后在项目页面中进行调用，会成功获取 github 返回的数据，但是一般我们在项目中配置接口的时候会直接省略项目 url 部分，比如：

```javascript
/* moduleA.js */
import Http from './http'

// 获取测试数据
export const getTestData = () => {
    return Http.get('/repos/octokit/octokit.rb')
}
```

这时候我们再次调用接口的时候会发现其调用地址为本地地址：`http://127.0.0.1:8080/repos/octokit/octokit.rb`，那么为了让其指向 `https://api.github.com`，我们需要在 vue.config.js 中进行 devServer 的配置：

```javascript
/* vue.config.js */

module.exports = {
    ...

    devServer: {

        // string | Object 代理设置
        proxy: {

            // 接口是 '/repos' 开头的才用代理
            '/repos': {
                target: 'https://api.github.com', // 目标地址
                changeOrigin: true, // 是否改变源地址
                // pathRewrite: {'^/api': ''}
            }
        },
    }

    ...
}
```

4. 公共设施配置

最后我们项目开发中肯定需要对一些公共的方法进行封装使用，这里我把它称之为公共设施，那么我们可以在 src 目录下建一个 common 文件夹来存放其配置文件：

```
└── src
    └── common
        ├── index.js      # 公共配置入口
        ├── validate.js   # 表单验证配置
        └── other.js      # 其他配置
```

在入口文件中我们可以向外暴露其他功能配置的模块，比如：

```javascript
/* index.js */
import Validate from './validate'
```

```
    Validate,
    Other,
}
```

这样我们在页面中只需要引入一个 index.js 即可。

结语

本文介绍了 Vue 单页应用的一些基本配置，从项目构建层面阐述了各文件的主要配置方式和注意点，由于本文并不是一篇文档类的配置说明，并不会详细介绍各配置文件的 API 功能，大家可以访问文中列出的官方文档进行查阅。

本案例代码地址：single-page-project

思考 & 作业

- devServer 中 proxy 的 key 值代表什么？如果再添加一个 `/reposed` 的配置会产生什么隐患？
- 如何配置 webpack 使得 `require.ensure()` 拆分出的 js 文件具有自定义文件名？

评论审核通过后显示 评论

MR.W

“如果是非线上环境，不加载VConsole”，说的这么绕口，而且有问题吧，不应该是：“非生产环境加载VConsole”?

▲ 0 收起评论 1月前

劳卜 前端工程师@ TC

嗯，使用“生产环境”更为贴切

21天前

评论审核通过后显示 评论

长依

请问接口封装完后，怎么使用呢，小白看不懂

▲ 0 评论 1月前

sonicsunsky

思考&作业没有答案？小白表示看不懂

▲ 0 评论 1月前

乐亦栗 前端

你好，请问let base = `${process.env.BASE_URL}` // 动态获取二级目录这里的process.env.BASE_URL 是什么呢？我该怎么设置呢?

▲ 0 收起评论 2月前

自

在vue.config.js 里面的baseUrl 设置

评论审核通过后显示 评论

苏本尊

配合实例看好一点，但是感觉并不透彻，实例也不完整，只是笔记。

▲ 5 评论 2月前

Salt同学

这个对小白的友好度太低了吧，新开一个项目都不从都开始搭建，配置。

▲ 1 收起评论 2月前

劳卜 前端工程师@ TC

也并非是新开，每一章都是循序渐进的过程，具体代码可以参考底部实例的

2月前

评论审核通过后显示 评论

雕刻时光本尊

使用以下两种方式： const Home = () => import(/* webpackChunkName: "about" */ './views/About.vue')或者const Home = resolve => { require.ensure(['./views/Home.vue'], () => { resolve(require('./views/Home.vue')) }) }... 展开全部

▲ 2 收起评论 2月前

劳卜 前端工程师@ TC

可以在讨论群加我，帮你看下

2月前

Yang /:sun

我也遇到相同的问题

2月前

Yang /:sun

clicked which looks like the lazy loading is working fine.

2月前

肖丫头 前端

回复 劳卜： 加群了,没同意我,哥

2天前

评论审核通过后显示 评论

®E·L·A·N© 会写点js @ 小猪短租

static isSuccess(res) { if (res.status >= 200 && res.status < 300) { return res } else { this.requestException(res) } }改成这样的写法： static isSuccess(res) {... 展开全部

▲ 0 收起评论 3月前

®E·L·A·N© 会写点js @ 小猪短租

for (var item in query) { let vals = query[item]; if (vals !== undefined) { params.push(item + '=' + query[item]) } } 既然都用ES6了这里用let of 或者object.entries转成数组后再用map、forEach 遍历会不会更好。看着ES5 和ES6混用，很奇怪。

3月前

®E·L·A·N© 会写点js @ 小猪短租

//获取测试数据export const getTestData = params => { return Http.get('https://api.github.com/repos/octokit/octokit.rb ') }这里没传params

3月前

劳卜 前端工程师@ TC

回复 ®E·L·A·N©： 嗯，ES6是ES5的补充，两者可以并存，优化了变量定义和无效参数

2月前

评论审核通过后显示 评论

是不是漏掉了哪一节？

▲ 0　收起评论　3月前

劳卜　前端工程师@ TC

创建的时候可以选择相应配置生成的

2月前

评论审核通过后显示　评论

钟xin科

//引入Home组件const Home = (resolve => { require.ensure(['./views/Home.vue'], () => { resolve(require('./views/Home.vue')) }) })应该改成： const Home = (resolve) => {...

展开全部

▲ 0　收起评论　3月前

劳卜　前端工程师@ TC

都可以，已优化书写

3月前

肖丫头　前端

es6,箭头函数,参数只有一个,可以不加括号的

2天前

评论审核通过后显示　评论

钟xin科

使用history模式，打包后router-view没有渲染出来，是为什么？

▲ 0　收起评论　3月前

劳卜　前端工程师@ TC

build 是生产环境的代码，资源读取的是相对于baseUrl 的地址，本地开发的话使用 npm run serve 预览就行了

评论审核通过后显示 评论

Si

1.第一个问题出现多个会有什么隐患? 2. 第二个问题在resolve 的第二个参数加文件名const Home = (resolve => { require.ensure(['./views/Home. vue'], () => { resolve(require('./views/Home.vue'), 'home') }) })...

▲ 0 评论 3月前

格子熊 web前端工程师@ 杭州某工地

您好，请问require.ensure()和import()有什么区别？我一直使用的是import()，也能异步加载。

▲ 0 收起评论 3月前

劳卜 前端工程师@ TC

实现原理不一样，作用基本相同，require.ensure 是webpack 特有的，遵循common.js 规范，而import 是ES6 的模块化引入方法，需要结合Vue 的异步组件才能实现按需加载

3月前

评论审核通过后显示 评论

SHERlocked93 前端打字员@ 中电

您好，路由配置这里的“如果路由存在二级目录，需要添加base 属性”这句话怎么理解呢，为什么存在二级目录就需要添加base属性

▲ 0 收起评论 3月前

劳卜 前端工程师@ TC

base 可以在所有路由前统一添加一个路径，避免重复书写二级目录名称

3月前

还有什么让我记住你 前端

评论审核通过后显示 评论

iwasmaster Front-End Developer @ Hikvision

"我们通过actions 提交mutations 去修改state 的值并通过getter 获取" 兄弟，这句话是不是有问题?

▲ 0 收起评论 3月前

劳卜 前端工程师@ TC

还请指出

3月前

iwasmaster Front-End Developer @ Hikvision

回复 劳卜： 用一句话是描述不清楚的。难道mutation都要靠action去提交？action提交的是mutaion没错，但一般只用于异步操作，通常是直接commit相应的mutaion就可以了，有很大的歧义。

3月前

劳卜 前端工程师@ TC

回复 iwasmaster： 嗯，这里主要为了说明它们之间的关系，已经优化了表述

3月前

评论审核通过后显示 评论

看今世繁华

请问require.ensure() 和component: () => import('@/views/About.vue') 作用一样么?

▲ 2 收起评论 3月前

萱萱爸爸爱搬砖 web前端开发@ 联想金融科技有限公司

同问

3月前

iwasmaster Front-End Developer @ Hikvision

一样的，推荐使用import() 官网原文：require.ensure() is specific to webpack and superseded by import()

3月前

iwasmaster Front-End Developer @ Hikvision

回复 萱萱爸爸爱搬砖： 一样的，推荐使用import()官网原文：require.ensure() is specific to webpack and superseded by import()

3月前

ALong

一样，据说是官方建议使用import（）方法

3月前

1 2 3 下一页

评论审核通过后显示

评论

hedeqiang PHP开发工程师@ 吉瑞德商

Nice

▲ 0 评论 3月前