# 构建实战篇4：项目整合与优化

前几小节，我们讲述了Vue 项目构建的整体流程，从无到有的实现了单页和多页应用的功能配置，但在实现的过程中不乏一些可以整合的功能点及可行性的优化方案，就像大楼造完需要进行最后的项目验收改进一样，有待我们进一步的去完善。

## 使用alias 简化路径

使用webpack构建过Vue项目的同学应该知道 `alias` 的作用，我们可以使用它将复杂的文件路径定义成一个变量来访问。在不使用alias的项目中，我们引入文件的时候通常会去计算被引入文件对于引入它的文件的相对路径，比如像这样：

```javascript
import HelloWorld from '../../../../HelloWorld.vue'
```

一旦相对层次结构较深，我们就很难去定位所引入文件的具体位置，其实这并不是我们应该操心的地方，完全可以交给webpack 来进行处理。在原生的webpack 配置中我们可以定义alias来解决这一问题：

```javascript
const path = require('path')

const resolve = dir => {
    return path.join(__dirname, dir)
}

module.exports = {
    ...

    resolve: {
        alias: {
            '@': resolve('src'), // 定义 src 目录变量
            _lib: resolve('src/common'), // 定义 common 目录变量,
            _com: resolve('src/components'), // 定义 components 目录变量,
```

```
    },

    ...
}
```

上方我们在webpack resolve（解析）对象下配置alias 的值，将常用的一些路径赋值给了我们自定义的变量，这样我们便可以将第一个例子简化为：

```javascript
import HelloWorld from '_com/HelloWorld.vue'
```

而在CLI 3.x 中我们无法直接操作webpack 的配置文件，我们需要通过chainWebpack 来进行间接修改，代码如下：

```javascript
/* vue.config.js */
module.exports = {
    ...

    chainWebpack: config => {
        config.resolve.alias
            .set('@', resolve('src'))
            .set('_lib', resolve('src/common'))
            .set('_com', resolve('src/components'))
            .set('_img', resolve('src/images'))
            .set('_ser', resolve('src/services'))
    },

    ...
}
```

这样我们修改webpack alias来简化路径的优化就实现了。但是需要注意的是对于在样式及html模板中引用路径的简写时，前面需要加上 ~ 符，否则路径解析会失败，如：

```css
.img {
    background: (~_img/home.png);
}
```

在多页应用的构建中，由于存在多个入口文件，因此会出现重复书写相同入口配置的情况，这样对于后期的修改和维护都不是特别友好，需要修改所有入口文件的相同配置，比如在index单页的入口中我们引用了VConsole 及performance 的配置，同时在Vue 实例上还添加了$openRouter 方法：

javascript

```
import Vue from 'vue'
import App from './index.vue'
import router from './router'
import store from '@/store/'
import { Navigator } from '../../common'

// 如果是非线上环境，不加载 VConsole
if (process.env.NODE_ENV !== 'production') {
    var VConsole = require('vconsole/dist/vconsole.min.js');
    var vConsole = new VConsole();

    Vue.config.performance = true;
}

Vue.$openRouter = Vue.prototype.$openRouter = Navigator.openRouter;

new Vue({
  router,
  store,
  render: h => h(App)
}).$mount('#app')
```

而在 page1 和 page2 的入口文件中也同样进行了上述配置，那我们该如何整合这些重复代码，使其能够实现一次修改多处生效的功能呢？最简单的方法便是封装成一个共用方法来进行调用，这里我们可以在 common 文件夹下新建 entryConfig 文件夹用于放置入口文件中公共配置的封装，封装代码如下：

javascript

```
import { Navigator } from '../index'

export default (Vue) => {

    // 如果是非线上环境，不加载 VConsole
    if (process.env.NODE_ENV !== 'production') {
        var VConsole = require('vconsole/dist/vconsole.min.js');
```

```
    }

    Vue.$openRouter = Vue.prototype.$openRouter = Navigator.openRouter;
}
```

上述代码我们向外暴露了一个函数，在调用它的入口文件中传入 Vue 实例作为参数即可实现内部功能的共用，我们可以将原本的入口文件简化为:

javascript

```
import Vue from 'vue'
import App from './index.vue'
import router from './router'
import store from '@/store/'
import entryConfig from '_lib/entryConfig/'

// 调用公共方法加载配置
entryConfig(Vue)

new Vue({
  router,
  store,
  render: h => h(App)
}).$mount('#app')
```

这样我们便完成了入口文件配置的整合，当然你还可以给该函数传入 router 实例及自定义参数用于其他共用配置的封装。

## 开启 Gzip 压缩

在《webpack 在 CLI 3 中的应用》章节，我们介绍了 CLI 为我们内置的 webpack plugins，使用这些内置插件基本已经能够满足我们大多数项目的构建和优化，当然你仍然可以为项目添加自己想要的插件来实现一些差异化的功能，比如使用 compression-webpack-plugin 来开启 Gzip 压缩。在 vue.config.js 配置文件中，我们通过 configureWebpack 中返回一个对象来实现 plugins 的合并：

javascript

```
/* vue.config.js */
const isPro = process.env.NODE_ENV === 'production'
```

```
    configureWebpack: config => {
        if (isPro) {
            return {
                plugins: [
                    new CompressionWebpackPlugin({
                         // 目标文件名称。[path] 被替换为原始文件的路径和 [query] 查询
                        asset: '[path].gz[query]',
                        // 使用 gzip 压缩
                        algorithm: 'gzip',
                        // 处理与此正则相匹配的所有文件
                        test: new RegExp(
                            '\\.(js|css)$'
                        ),
                        // 只处理大于此大小的文件
                        threshold: 10240,
                        // 最小压缩比达到 0.8 时才会被压缩
                        minRatio: 0.8,
                    })
                ]
            }
        }
        ...
}
```

上方我们通过在生产环境中增加 Gzip 压缩配置实现了打包后输出增加对应的 .gz 为后缀的文件，而由于我们配置项中配置的是只压缩大小超过 10240B（10kB）的 JS 及 CSS，因此不满足条件的文件不会进行 Gzip 压缩。

Gzip 压缩能在普通压缩的基础上再进行 50% 以上 的压缩，我们可以直接来看下控制台的输出对比图：

很明显，Gzip 压缩后的文件体积得到了很大程度的减小，这对于浏览器资源加载速度的提升起到了非常有效的帮助。但是需要注意的是访问 Gzip 压缩的文件需要服务端进行相应配置，以下是 Nginx Gzip 压缩的流程：

> Nginx 开启 Gzip 压缩配置后，其会根据配置情况对指定的类型文件进行压缩，主要针对 JS 与 CSS 。如果文件路径中存在与原文件同名（加了个 .gz），Nginx 会获取 gz 文件，如果找不到，会主动进行 Gzip 压缩。

## 结语

至此，一路走来，我们成功完成了本小册 Vue 项目构建部分的教程，从 CLI 3.x 的使用到项目内外部环境的配置，再到最后多页应用的拓展，我们循序渐进、由浅入深的讲解了 Vue 项目构建的主要知识点及详细流程，希望大家能够在此基础上举一反三，结合实际代码，将理论知识转化为实际运用，配合自己的理解，一步步实现自己的项目构建，并为构建出的项目添砖加瓦，实现质的飞跃。

## 思考 & 作业

- 除了本文中介绍的项目优化方法，还有哪些常见的优化手段？如何通过 Vue CLI 3 配置实现？

## 留言

评论将在后台进行审核，审核通过后对所有人可见

**jhhost**

“访问 Gzip 压缩的文件需要服务端进行相应配置“ 。那服务端该如何配置?

▲ 0　　评论　7天前

**Dong_**

“访问 Gzip 压缩的文件需要服务端进行相应配置“ 。那服务端该如何配置?

▲ 0　　评论　1月前

**雕刻时光本尊**

请教作者：多页应用MPA中，vuex配置是不是在src/pages/下每一个页面的目录下增加个store，这样是比较优雅的结构吗？每个页面的store并不能共享吧?

▲ 0　　收起评论　2月前

雕刻时光本尊

我看MPA示例中，是在src下有一个全局统一的store目录，用于全局状态管理。请问这种适合MPA吗？我记得vuex中的数据，在页面刷新后就丢失了。

2月前

劳卜　前端工程师 @ TC

多页间 store 不共享的，可以每个单页单独一个 store

2月前

评论审核通过后显示　　评论

▲ 0　收起评论　3月前

chenzesam 前端 @ 搬砖

base64 小图标

3月前

评论审核通过后显示　评论

**MonkeyChen** 前端

compression-webpack-plugin v.2.0.0 中，配置项中的 `asset` 应改为 `filename`，否则将报以下错误：

ERROR ValidationError: Compression Plugin Invalid Options

options should NOT have additional properties

ValidationError: Compression Plugin Invalid Options... 展开全部

▲ 0　收起评论　3月前

劳卜 前端工程师 @ TC

感谢提醒，使用 2.0 版本的可以注意一下

3月前

雕刻时光本尊

asset改成了filename，接口有改变。

2月前

评论审核通过后显示　评论