# 网站子目录部署VUE webpack 打包资源文件路径的正确引用方式

webpack 是目前使用最为火热的打包工具，各大知名的框架类库都用其打包，国内使用最近也火热起来。它在单页应用和类库打包上帮助许多人从代码管理中解脱了出来，成为了当下风靡一时的打包工具。

但是坑也很多，比如说图片，字体等文件的路径。

刚开始用webpack的同学很容易掉进图片打包这个坑里，比如打包出来的图片地址不对或者有的图片并不能打包进我们的目标文件夹里。明明在开发阶段都是好好的，一但发布到线上，就出现各种404，

一般来说，webpack打包的SPA程序，发布到网站的根目录下都不会出现太多问题，但是发布到网站的子目下，就会出现各种资源文件找不到的情况。

这种文件，我在以下这个知乎的帖子里曾经做过详细的回答

知乎用户：vuejs怎么在服务器部署?

似乎所有问题都解决了，但有一个问题没有解决，就是如果我在css里引了了图片资源，webpack并不能很好的处理这里面的资源路径

比如我把spa部署在 https://www.wx2share.com/m/
“m”文件夹是网站下的一个子目录，

如果我在开发的时候，写了如下css代码

```
.content {
  background: url('/static/img/1.jpg')
}
```

首先，我们修改(tips:我的配置文件是通过vue-cli生成的)

```
build: {
        env: require('./prod.env'),
        index: path.resolve(__dirname, '../dist/index.html'),
```

```
        assetsRoot: path.resolve(__dirname, '../dist'),
        assetsSubDirectory: 'static',
        assetsPublicPath: '/m/', //这里指定publicPath 的路径为我子目录文件名，一
        productionSourceMap: true,

    },
```

webpack publicPath 参数是用来，帮助你为项目中的所有资源指定一个基础路径，一但设定值以后，所有代码中通过requrie 或import 方式引入的资源文件，在build以后，都回指向类似 '/m/static/xxx.xx'

比如：

```
logo: require('@/assets/v.png')

<img :src="logo">
```

编译以后，

```
<div class="logo-warp">
<img src="/m/static/img/logo_small.f457a3f.png">
</div>
```

可以看到，webpack 已经正确的把资源文件的路径里 加上了/m ，保证了资源文件引用正确，这里顺便提一点，publicpath 还可以设置在cdn的url

```
如果 publicPath: 'https://cdn.youdomin.com/' 这样编译以后，src=''https://cdn.y
```

这样你只要把static整个文件夹托管的你的cdn上就可以了非常方便。但是 publicPath **一定用绝对路径，绝对路径，绝对路径（重要的事说3遍）** 千万不要用相对路径，如果你把publicPath设为'./',哪你启用路由以后，https://www.wx2share.com/m/ 这样的url进入程序的，不会有问题，因为这个时候，static文件正好在当前目录下，但是当你类似用这样的网址来访问的时候 ，https://www.wx2share.com/m/sh... 资源文件又找不到了，因为这个时候./的指向的目录是

/m/show/static/ 很明显你的资源文件全在 /m/static下，所以又404了。

上面的方法基本解决了大部分的问题，唯一不能解决的就是文章开头提到的，css中引入的资源文件的问题了

通过publicpath的设置，并不到改变css中引用资源文件的路径，上面实例代码中的css编译后，还是

```
.content {
  background: url('/static/img/1.jpg')
}
```

是乎webpack 并不处理css中的文件路径，这样的结果就是发布以后页面上所有通过css引入的资源文件全部不能正常显示了。前段时间我采用了一个简单又粗爆的方法来解决这个问题，哪就是，绝对不用css引入任何文件。当然你也可以手动修改webpack编译过的文件，把通过查找替换 把/static 替换为/m/static 如果你不嫌累的慌。

最近又开始了一个新的SPA应用，引入了一个第三方css，里面好多通过css引入的图片文件 ，让我不得不下定决心来解决这个问题了

TIPS: 以下内容的配置文件修改全部是基于由vue-cli生成的配置文件，如果你是用自己写的配置文件请注意区别!

我首先想到，如果开发的时候我就是指定在“m/”子目录下，不就能解决大部分问题了，这样和线上的根目录文件对应了，buid以后就不会有哪么多问题，

第一步：修改dev模式上的publicPath

```
dev: {
        env: require('./dev.env'),
        port: 8082,
        autoOpenBrowser: true,
        assetsSubDirectory: 'static',
        assetsPublicPath: '/m/', //把dev模式下的publicPath也设为 m
        proxyTable: {},

        cssSourceMap: false
    }
```

**yarn run dev**

以后，访问 http://localhost:8082/m/ 结果好么，直接给我来了个 can not get /m/

但是如果我直接访问http://localhost:8082/m/index.html 是可以的，而且功能基本正常，是就vue的router不起作用了，在浏览器地址栏里直接敲入 http://loalhost:8082/m/view/1 然后回车，这样的地址就全部404了，看来是dev sever没有把，所有链接从新定向到 /m/index.html上，但是publicPath设置是正确的了，现在要解决的问题就是，无论在浏览器地址栏里输入什么网址，都让它重定向到 /m/index.html 就可以解决所有问题了，

webpack 的dev sever 是什么，用什么来实现的，能过查看buid/dev-server.js

发现是用express 来实现web server,通过加载webpack-dev-middleware 来实现实时编译，所有请求都转发给它了，所以，在node_modules下找到它的源代码，看了老半天，看不出所以然来，只能debug了，看它到底是怎么运作的，电脑上没有调试的工具，只好，通过打log的方法来调试了。

```
function webpackDevMiddleware(req, res, next) {
        function goNext() {
            if(!context.options.serverSideRender) return next();
            return new Promise(function(resolve) {
                shared.ready(function() {
                    res.locals.webpackStats = context.webpackStats;
                    resolve(next());
                }, req);
            });
        }

        if(req.method !== "GET") {
            return goNext();
        }

        var filename = getFilenameFromUrl(context.options.publicPath, conte
                console.log(filename) //关键就是这里，只有filename不等于 false的
        if(filename === false) return goNext();
        //下面还有好多代码，不粘贴了，
```

既然这里返回 false

我们进入 getFileNameFromUrl 这个函数看看，为什么会false

```
function getFilenameFromUrl(publicPath, outputPath, url) {
```

```
var filename;
console.log(publicPath, outputPath, url) //我在这里打了个log
// localPrefix is the folder our bundle should be in
var localPrefix = urlParse(publicPath || "/", false, true);
var urlObject = urlParse(url);

// publicPath has the hostname that is not the same as request url's, s
if(localPrefix.hostname !== null && urlObject.hostname !== null &&
    localPrefix.hostname !== urlObject.hostname) {
    return false;
}

// publicPath is not in url, so it should fail
if(publicPath && localPrefix.hostname === urlObject.hostname && url.ind
    return false; //就这里return false了
}

// strip localPrefix from the start of url
if(urlObject.pathname.indexOf(localPrefix.pathname) === 0) {
    filename = urlObject.pathname.substr(localPrefix.pathname.length);
}

if(!urlObject.hostname && localPrefix.hostname &&
    url.indexOf(localPrefix.path) !== 0) {
        return false;
    }
    // and if not match, use outputPath as filename
    return querystring.unescape(filename ? pathJoin(outputPath, filename) :
}
```

看上面代码，我在进入函数的头部打了一个log，看看传入的参数到底是什么

> 当我访问 http://localhost:8082/m/ 的时候 控制台里输出
>
> 发现 publicPath, outputPath, url，三个参数的值分别为：
>
> /m/ F:workspacewx2share-pwadist /index.html

终于发现

if(publicPath && localPrefix.hostname === urlObject.hostname &&
url.indexOf(publicPath) !== 0) {

```
    return false; //就这里return false了
}
```

url.index.of(publicPath) !== 0 这一个条件成立了 相当于 '/index.html/'.indexOf('/m/') 肯定不会===0 啊，

但是这个‘/index.html' 这个参数的值哪里来的，回到上一段代码中，发现是req.url里传过来的，

但我明明访问的是/m/ 哪，req.url 应该等于 '/m/'啊，这是什么时候给重定向的呢，看来在这个之前，已经有过一次重定向了

回到dev-server.js文件，发现在，use devMiddlewarre 之前，还引入了一个connect-history-api-fallback的中间件，看来唯一能重定向的地方只有这里了，

```
app.use(require('connect-history-api-fallback')());

// 服务器部署 webpack 打包的静态资源
app.use(devMiddleware);

// 使用热更新， 如果编译出现错误会实时展示编译错误
app.use(hotMiddleware);
```

打开connect-history-api-fallback的代码，终于发现了

```
rewriteTarget = options.index || '/index.html';
    logger('Rewriting', req.method, req.url, 'to', rewriteTarget);
    req.url = rewriteTarget;
    next();
```

这个的代码，当你不作配置的时候，默认重定向到根目录的/index.html上，找到原因了，修改就简

```
// 处理 history API 的回退情况（如果在线上环境中，也需要服务器做相应处理）
app.use(require('connect-history-api-fallback')({
    index: '/m/index.html'
}));
```

把options.index的值设为 /m/index.html 不就可以了吧，这样所有的请求，都会转发到/m/index.html了

再次打开http://localhost:8082/m/ 一切全部正常了，build以后，发现在线上，也完全正常，再也不会找不到css中引入的资源文件了

一个重点差点忘记提了

就是现在写css代码里，引入static文件中的文件，直接要写加上publicPath的路径，

就是原来写成

```
.content {
  background: url('/static/img/1.jpg')
}
```

要在编码阶段全部写成

```
.content {
  background: url('/m/static/img/1.jpg')  //直接带上发布是的绝对路径
}
```