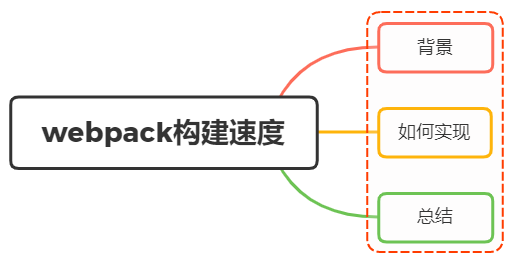
# 面试官：如何提高webpack的构建速度？



## 一、背景

随着我们的项目涉及到页面越来越多，功能和业务代码也会随着越多，相应的 webpack 的构建时间也会越来越久

构建时间与我们日常开发效率密切相关，当我们本地开发启动 devServer 或者 build 的时候，如果时间过长，会大大降低我们的工作效率

所以，优化webpack 构建速度是十分重要的环节

## 二、如何优化

常见的提升构建速度的手段有如下：

* 优化 loader 配置
* 合理使用 resolve.extensions
* 优化 resolve.modules
* 优化 resolve.alias
* 使用 DLLPlugin 插件
* 使用 cache-loader
* terser 启动多线程
* 合理使用 sourceMap

### 优化loader配置

在使用loader时，可以通过配置include、exclude、test属性来匹配文件，接触include、exclude规定哪些匹配应用loader

如采用 ES6 的项目为例，在配置 babel-loader时，可以这样：

module.exports = {  
 module: {  
 rules: [  
 {  
 // 如果项目源码中只有 js 文件就不要写成 /\.jsx?$/，提升正则表达式性能  
 test: /\.js$/,  
 // babel-loader 支持缓存转换出的结果，通过 cacheDirectory 选项开启  
 use: ['babel-loader?cacheDirectory'],  
 // 只对项目根目录下的 src 目录中的文件采用 babel-loader  
 include: path.resolve(\_\_dirname, 'src'),  
 },  
 ]  
 },  
};

### 合理使用 resolve.extensions

在开发中我们会有各种各样的模块依赖，这些模块可能来自于自己编写的代码，也可能来自第三方库， resolve可以帮助webpack从每个 require/import 语句中，找到需要引入到合适的模块代码

通过resolve.extensions是解析到文件时自动添加拓展名，默认情况如下：

module.exports = {  
 ...  
 extensions:[".warm",".mjs",".js",".json"]  
}

当我们引入文件的时候，若没有文件后缀名，则会根据数组内的值依次查找

当我们配置的时候，则不要随便把所有后缀都写在里面，这会调用多次文件的查找，这样就会减慢打包速度

### 优化 resolve.modules

resolve.modules 用于配置 webpack 去哪些目录下寻找第三方模块。默认值为['node\_modules']，所以默认会从node\_modules中查找文件 当安装的第三方模块都放在项目根目录下的 ./node\_modules目录下时，所以可以指明存放第三方模块的绝对路径，以减少寻找，配置如下：

module.exports = {  
 resolve: {  
 // 使用绝对路径指明第三方模块存放的位置，以减少搜索步骤  
 // 其中 \_\_dirname 表示当前工作目录，也就是项目根目录  
 modules: [path.resolve(\_\_dirname, 'node\_modules')]  
 },  
};

### 优化 resolve.alias

alias给一些常用的路径起一个别名，特别当我们的项目目录结构比较深的时候，一个文件的路径可能是./../../的形式

通过配置alias以减少查找过程

module.exports = {  
 ...  
 resolve:{  
 alias:{  
 "@":path.resolve(\_\_dirname,'./src')  
 }  
 }  
}

### 使用 DLLPlugin 插件

DLL全称是 动态链接库，是为软件在winodw种实现共享函数库的一种实现方式，而Webpack也内置了DLL的功能，为的就是可以共享，不经常改变的代码，抽成一个共享的库。这个库在之后的编译过程中，会被引入到其他项目的代码中

使用步骤分成两部分：

* 打包一个 DLL 库
* 引入 DLL 库

#### 打包一个 DLL 库

webpack内置了一个DllPlugin可以帮助我们打包一个DLL的库文件

module.exports = {  
 ...  
 plugins:[  
 new webpack.DllPlugin({  
 name:'dll\_[name]',  
 path:path.resolve(\_\_dirname,"./dll/[name].mainfest.json")  
 })  
 ]  
}

#### 引入 DLL 库

使用 webpack 自带的 DllReferencePlugin 插件对 mainfest.json 映射文件进行分析，获取要使用的DLL库

然后再通过AddAssetHtmlPlugin插件，将我们打包的DLL库引入到Html模块中

module.exports = {  
 ...  
 new webpack.DllReferencePlugin({  
 context:path.resolve(\_\_dirname,"./dll/dll\_react.js"),  
 mainfest:path.resolve(\_\_dirname,"./dll/react.mainfest.json")  
 }),  
 new AddAssetHtmlPlugin({  
 outputPath:"./auto",  
 filepath:path.resolve(\_\_dirname,"./dll/dll\_react.js")  
 })  
}

### 使用 cache-loader

在一些性能开销较大的 loader之前添加 cache-loader，以将结果缓存到磁盘里，显著提升二次构建速度

保存和读取这些缓存文件会有一些时间开销，所以请只对性能开销较大的 loader 使用此loader

module.exports = {  
 module: {  
 rules: [  
 {  
 test: /\.ext$/,  
 use: ['cache-loader', ...loaders],  
 include: path.resolve('src'),  
 },  
 ],  
 },  
};

### terser 启动多线程

使用多进程并行运行来提高构建速度

module.exports = {  
 optimization: {  
 minimizer: [  
 new TerserPlugin({  
 parallel: true,  
 }),  
 ],  
 },  
};

### 合理使用 sourceMap

打包生成 sourceMap 的时候，如果信息越详细，打包速度就会越慢。对应属性取值如下所示：



### 三、总结

可以看到，优化webpack构建的方式有很多，主要可以从优化搜索时间、缩小文件搜索范围、减少不必要的编译等方面入手

## 参考文献

* https://github.com/ly2011/blog/issues/44
* https://xie.infoq.cn/article/541418eb82a674741a0ad8865
* https://zhuanlan.zhihu.com/p/139498741
* https://vue3js.cn/interview