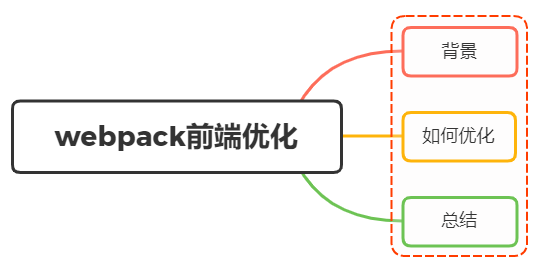
# 面试官：说说如何借助webpack来优化前端性能？



## 一、背景

随着前端的项目逐渐扩大，必然会带来的一个问题就是性能

尤其在大型复杂的项目中，前端业务可能因为一个小小的数据依赖，导致整个页面卡顿甚至奔溃

一般项目在完成后，会通过webpack进行打包，利用webpack对前端项目性能优化是一个十分重要的环节

## 二、如何优化

通过webpack优化前端的手段有：

* JS代码压缩
* CSS代码压缩
* Html文件代码压缩
* 文件大小压缩
* 图片压缩
* Tree Shaking
* 代码分离
* 内联 chunk

### JS代码压缩

terser是一个JavaScript的解释、绞肉机、压缩机的工具集，可以帮助我们压缩、丑化我们的代码，让bundle更小

在production模式下，webpack 默认就是使用 TerserPlugin 来处理我们的代码的。如果想要自定义配置它，配置方法如下：

const TerserPlugin = require('terser-webpack-plugin')  
module.exports = {  
 ...  
 optimization: {  
 minimize: true,  
 minimizer: [  
 new TerserPlugin({  
 parallel: true // 电脑cpu核数-1  
 })  
 ]  
 }  
}

属性介绍如下：

* extractComments：默认值为true，表示会将注释抽取到一个单独的文件中，开发阶段，我们可设置为 false ，不保留注释
* parallel：使用多进程并发运行提高构建的速度，默认值是true，并发运行的默认数量： os.cpus().length - 1
* terserOptions：设置我们的terser相关的配置：
* compress：设置压缩相关的选项，mangle：设置丑化相关的选项，可以直接设置为true
* mangle：设置丑化相关的选项，可以直接设置为true
* toplevel：底层变量是否进行转换
* keep\_classnames：保留类的名称
* keep\_fnames：保留函数的名称

### CSS代码压缩

CSS压缩通常是去除无用的空格等，因为很难去修改选择器、属性的名称、值等

CSS的压缩我们可以使用另外一个插件：css-minimizer-webpack-plugin

npm install css-minimizer-webpack-plugin -D

配置方法如下：

const CssMinimizerPlugin = require('css-minimizer-webpack-plugin')  
module.exports = {  
 // ...  
 optimization: {  
 minimize: true,  
 minimizer: [  
 new CssMinimizerPlugin({  
 parallel: true  
 })  
 ]  
 }  
}

### Html文件代码压缩

使用HtmlWebpackPlugin插件来生成HTML的模板时候，通过配置属性minify进行html优化

module.exports = {  
 ...  
 plugin:[  
 new HtmlwebpackPlugin({  
 ...  
 minify:{  
 minifyCSS:false, // 是否压缩css  
 collapseWhitespace:false, // 是否折叠空格  
 removeComments:true // 是否移除注释  
 }  
 })  
 ]  
}

设置了minify，实际会使用另一个插件html-minifier-terser

### 文件大小压缩

对文件的大小进行压缩，减少http传输过程中宽带的损耗

npm install compression-webpack-plugin -D

new ComepressionPlugin({  
 test:/\.(css|js)$/, // 哪些文件需要压缩  
 threshold:500, // 设置文件多大开始压缩  
 minRatio:0.7, // 至少压缩的比例  
 algorithm:"gzip", // 采用的压缩算法  
})

### 图片压缩

一般来说在打包之后，一些图片文件的大小是远远要比 js 或者 css 文件要来的大，所以图片压缩较为重要

配置方法如下：

module: {  
 rules: [  
 {  
 test: /\.(png|jpg|gif)$/,  
 use: [  
 {  
 loader: 'file-loader',  
 options: {  
 name: '[name]\_[hash].[ext]',  
 outputPath: 'images/',  
 }  
 },  
 {  
 loader: 'image-webpack-loader',  
 options: {  
 // 压缩 jpeg 的配置  
 mozjpeg: {  
 progressive: true,  
 quality: 65  
 },  
 // 使用 imagemin\*\*-optipng 压缩 png，enable: false 为关闭  
 optipng: {  
 enabled: false,  
 },  
 // 使用 imagemin-pngquant 压缩 png  
 pngquant: {  
 quality: '65-90',  
 speed: 4  
 },  
 // 压缩 gif 的配置  
 gifsicle: {  
 interlaced: false,  
 },  
 // 开启 webp，会把 jpg 和 png 图片压缩为 webp 格式  
 webp: {  
 quality: 75  
 }  
 }  
 }  
 ]  
 },  
 ]  
}

### Tree Shaking

Tree Shaking 是一个术语，在计算机中表示消除死代码，依赖于ES Module的静态语法分析（不执行任何的代码，可以明确知道模块的依赖关系）

在webpack实现Trss shaking有两种不同的方案：

* usedExports：通过标记某些函数是否被使用，之后通过Terser来进行优化的
* sideEffects：跳过整个模块/文件，直接查看该文件是否有副作用

两种不同的配置方案， 有不同的效果

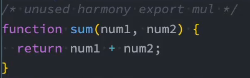
#### usedExports

配置方法也很简单，只需要将usedExports设为true

module.exports = {  
 ...  
 optimization:{  
 usedExports  
 }  
}

使用之后，没被用上的代码在webpack打包中会加入unused harmony export mul注释，用来告知 Terser 在优化时，可以删除掉这段代码

如下面sum函数没被用到，webpack打包会添加注释，terser在优化时，则将该函数去掉



#### sideEffects

sideEffects用于告知webpack compiler哪些模块时有副作用，配置方法是在package.json中设置sideEffects属性

如果sideEffects设置为false，就是告知webpack可以安全的删除未用到的exports

如果有些文件需要保留，可以设置为数组的形式

"sideEffecis":[  
 "./src/util/format.js",  
 "\*.css" // 所有的css文件  
]

上述都是关于javascript的tree shaking，css同样也能够实现tree shaking

#### css tree shaking

css进行tree shaking优化可以安装PurgeCss插件

npm install purgecss-plugin-webpack -D

const PurgeCssPlugin = require('purgecss-webpack-plugin')  
module.exports = {  
 ...  
 plugins:[  
 new PurgeCssPlugin({  
 path:glob.sync(`${path.resolve('./src')}/\*\*/\*`), {nodir:true}// src里面的所有文件  
 satelist:function(){  
 return {  
 standard:["html"]  
 }  
 }  
 })  
 ]  
}

* paths：表示要检测哪些目录下的内容需要被分析，配合使用glob
* 默认情况下，Purgecss会将我们的html标签的样式移除掉，如果我们希望保留，可以添加一个safelist的属性

### 代码分离

将代码分离到不同的bundle中，之后我们可以按需加载，或者并行加载这些文件

默认情况下，所有的JavaScript代码（业务代码、第三方依赖、暂时没有用到的模块）在首页全部都加载，就会影响首页的加载速度

代码分离可以分出出更小的bundle，以及控制资源加载优先级，提供代码的加载性能

这里通过splitChunksPlugin来实现，该插件webpack已经默认安装和集成，只需要配置即可

默认配置中，chunks仅仅针对于异步（async）请求，我们可以设置为initial或者all

module.exports = {  
 ...  
 optimization:{  
 splitChunks:{  
 chunks:"all"  
 }  
 }  
}

splitChunks主要属性有如下：

* Chunks，对同步代码还是异步代码进行处理
* minSize： 拆分包的大小, 至少为minSize，如何包的大小不超过minSize，这个包不会拆分
* maxSize： 将大于maxSize的包，拆分为不小于minSize的包
* minChunks：被引入的次数，默认是1

### 内联chunk

可以通过InlineChunkHtmlPlugin插件将一些chunk的模块内联到html，如runtime的代码（对模块进行解析、加载、模块信息相关的代码），代码量并不大，但是必须加载的

const InlineChunkHtmlPlugin = require('react-dev-utils/InlineChunkHtmlPlugin')  
const HtmlWebpackPlugin = require('html-webpack-plugin')  
module.exports = {  
 ...  
 plugin:[  
 new InlineChunkHtmlPlugin(HtmlWebpackPlugin,[/runtime.+\.js/]  
}

### 三、总结

关于webpack对前端性能的优化，可以通过文件体积大小入手，其次还可通过分包的形式、减少http请求次数等方式，实现对前端性能的优化

## 参考文献

* https://zhuanlan.zhihu.com/p/139498741
* https://vue3js.cn/interview/