---

link: null
title: 珠峰架构师成长计划
description: webpack.config.js
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=65 sentences=200, words=1563

---

## 1. 提取CSS #

- 因为CSS的下载和JS可以并行,当一个HTML文件很大的时候，我们可以把CSS单独提取出来加载

### 1.1 安装 #

- mini-css-extract-plugin (https://github.com/webpack-contrib/mini-css-extract-plugin)

```
npm install  mini-css-extract-plugin --save-dev
```

### 1.2 webpack.config.js #

webpack.config.js

```
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');
+const MiniCssExtractPlugin = require('mini-css-extract-plugin');
module.exports = {
  mode: 'development',
  devtool: false,
  entry: './src/index.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: '[name].js',
+    publicPath: '/'
  },
  module: {
    rules: [
      { test: /\.txt$/, use: 'raw-loader' },
+      { test: /\.css$/, use: [MiniCssExtractPlugin.loader, 'css-loader'] },
+      { test: /\.less$/, use: [MiniCssExtractPlugin.loader, 'css-loader', 'less-loader'] },
+      { test: /\.scss$/, use: [MiniCssExtractPlugin.loader, 'css-loader', 'sass-loader'] },
       {
        test: /\.(jpg|png|gif|bmp|svg)$/,
        type:'asset/resource',
        generator:{
          filename:'images/[hash][ext]'
        }
       }
    ]
  },
  plugins: [
    new HtmlWebpackPlugin({ template: './src/index.html' }),
+    new MiniCssExtractPlugin({
+      filename: '[name].css'
+    })
  ]
};
```

## 2 指定图片和CSS目录 #

### 2.1 webpack.config.js #

```
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
module.exports = {
  mode: 'development',
  devtool: false,
  entry: './src/index.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: '[name].js',
    publicPath: '/'
  },
  module: {
    rules: [
      { test: /\.txt$/, use: 'raw-loader' },
      { test: /\.css$/, use: [MiniCssExtractPlugin.loader, 'css-loader'] },
      { test: /\.less$/, use: [MiniCssExtractPlugin.loader, 'css-loader', 'less-loader'] },
      { test: /\.scss$/, use: [MiniCssExtractPlugin.loader, 'css-loader', 'sass-loader'] },
      {
        test: /\.(jpg|png|gif|bmp|svg)$/,
        type:'asset/resource',
        generator:{
          filename:'images/[hash][ext]'
        }
      }
    ]
  },
  plugins: [
    new HtmlWebpackPlugin({ template: './src/index.html' }),
    new MiniCssExtractPlugin({
+      filename: 'css/[name].css'
    }),
  ]
};
```

### 3. 压缩JS、CSS和HTML #

- optimize-css-assets-webpack-plugin (https://www.npmjs.com/package/optimize-css-assets-webpack-plugin)是一个优化和压缩CSS资源的插件
- terser-webpack-plugin (https://www.npmjs.com/package/terser-webpack-plugin)是一个优化和压缩JS资源的插件

webpack.config.js

```
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
+const OptimizeCssAssetsWebpackPlugin = require('optimize-css-assets-webpack-plugin');
+const TerserPlugin = require('terser-webpack-plugin');

module.exports = {
+  mode: 'none',
  devtool: false,
  entry: './src/index.js',
+  optimization: {
+    minimize: true,
+    minimizer: [
+      new TerserPlugin(),
+    ],
+  },
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: '[name].js',
    publicPath: '/',
  },
  devServer: {
    contentBase: path.resolve(__dirname, 'dist'),
    compress: true,
    port: 8080,
    open: true,
  },
  module: {
    rules: [
      {
        test: /\.jsx?$/,
        loader: 'eslint-loader',
        enforce: 'pre',
        options: { fix: true },
        exclude: /node_modules/,
      },
      {
        test: /\.jsx?$/,
        use: {
          loader: 'babel-loader',
          options: {
            presets: [[
              '@babel/preset-env',
              {
                useBuiltIns: 'usage'
                corejs: {
                  version: 3
                },
                targets: {
                  chrome: '60',
                  firefox: '60',
                  ie: '9',
                  safari: '10',
                  edge: '17',
                },
              },
            ], '@babel/preset-react'],
            plugins: [
              ['@babel/plugin-proposal-decorators', { legacy: true }],
              ['@babel/plugin-proposal-class-properties', { loose: true }],
            ],
          },
        },
        include: path.join(__dirname, 'src'),
        exclude: /node_modules/,
      },
      { test: /\.txt$/, use: 'raw-loader' },
      { test: /\.css$/, use: [MiniCssExtractPlugin.loader, 'css-loader', 'postcss-loader'] },
      { test: /\.less$/, use: [MiniCssExtractPlugin.loader, 'css-loader', 'postcss-loader', 'less-loader'] },
      { test: /\.scss$/, use: [MiniCssExtractPlugin.loader, 'css-loader', 'postcss-loader', 'sass-loader'] },
      {
        test: /\.(jpg|png|gif|bmp|svg)$/,
        type:'asset/resource',
        generator:{
          filename:'images/[hash][ext]'
        }
      },
      {
        test: /\.html$/,
        loader: 'html-loader',
      },
    ],
  },
  plugins: [
    new HtmlWebpackPlugin({
    template: './src/index.html',
+     minify: {
+        collapseWhitespace: true,
+        removeComments: true
      }
    }),
    new MiniCssExtractPlugin({
      filename: 'css/[name].css',
    }),
+    new OptimizeCssAssetsWebpackPlugin(),
  ],
};
```

### 4. CDN #

- qiniu (https://www.qiniu.com/)

- CDN 又叫内容分发网络，通过把资源部署到世界各地，用户在访问时按照就近原则从离用户最近的服务器获取资源，从而加速资源的获取速度。
- public-path (https://webpack.js.org/guides/public-path/#root)
- external-remotes-plugin (https://npmmirror.com/package/external-remotes-plugin)


### 4.1 使用缓存 #

- HTML文件不缓存，放在自己的服务器上，关闭自己服务器的缓存，静态资源的URL变成指向CDN服务器的地址
- 静态的JavaScript、CSS、图片等文件开启CDN和缓存，并且文件名带上HASH值
- 为了并行加载不阻塞，把不同的静态资源分配到不同的CDN服务器上

### 4.2 域名限制 #

- 同一时刻针对同一个域名的资源并行请求是有限制
- 可以把这些静态资源分散到不同的 CDN 服务上去
- 多个域名后会增加域名解析时间
- 可以通过在 HTML HEAD 标签中 加入 `<link rel="dns-prefetch" href="http://img.zhufengpeixun.cn">` 去预解析域名，以降低域名解析带来的延迟

### 4.3 文件指纹 #

- 打包后输出的文件名和后缀
- hash一般是结合CDN缓存来使用，通过webpack构建之后，生成对应文件名自动带上对应的MD5值。如果文件内容改变的话，那么对应文件哈希值也会改变，对应的HTML引用的URL地址也会改变，触发CDN服务器从源服务器上拉取对应数据，进而更新本地缓存。

指纹占位符

占位符名称 含义 ext 资源后缀名 name 文件名称 path 文件的相对路径 folder 文件所在的文件夹 hash 每次webpack构建时生成一个唯一的hash值 chunkhash 根据chunk生成hash值，来源于同一个chunk，则hash值就一样 contenthash 根据内容生成hash值，文件内容相同hash值就相同

#### 4.3.1 hash #

- Hash 是整个项目的hash值，其根据每次编译内容计算得到，每次编译之后都会生成新的hash,即修改任何文件都会导致所有文件的hash发生改变

```
const path = require("path");
const glob = require("glob");
const PurgecssPlugin = require("purgecss-webpack-plugin");
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
const PATHS = {
  src: path.join(__dirname, 'src')
}
module.exports = {
  mode: "production",
+  entry: {
+    main: './src/index.js',
+    vender:['lodash']
+  },
  output:{
    path:path.resolve(__dirname,'dist'),
+    filename:'[name].[hash].js'
  },
  devServer:{
    hot:false
  },
  module: {
    rules: [
      {
        test: /\.js/,
        include: path.resolve(__dirname, "src"),
        use: [
          {
            loader:'thread-loader',
            options:{
              workers:3
            }
          },
          {
            loader: "babel-loader",
            options: {
              presets: ["@babel/preset-env", "@babel/preset-react"],
            },
          },
        ],
      },
      {
        test: /\.css$/,
        include: path.resolve(__dirname, "src"),
        exclude: /node_modules/,
        use: [
          {
            loader: MiniCssExtractPlugin.loader,
          },
          "css-loader",
        ],
      },
    ],
  },
  plugins: [
    new MiniCssExtractPlugin({
+      filename: "[name].[hash].css"
    }),
    new PurgecssPlugin({
      paths: glob.sync(`${PATHS.src}/**/*`,  { nodir: true }),
    }),
  ],
};
```

#### 4.3.2 chunkhash #

- chunkhash 采用hash计算的话，每一次构建后生成的哈希值都不一样，即使文件内容压根没有改变。这样子是没办法实现缓存效果，我们需要换另一种哈希值计算方式，即chunkhash,chunkhash和hash不一样，它根据不同的入口文件(Entry)进行依赖文件解析、构建对应的chunk，生成对应的哈希值。我们在生产环境里把一些公共库和程序入口文件区分开，单独打包构建，接着我们采用chunkhash的方式生成哈希值，那么只要我们不改动公共库的代码，就可以保证其哈希值不会受影响

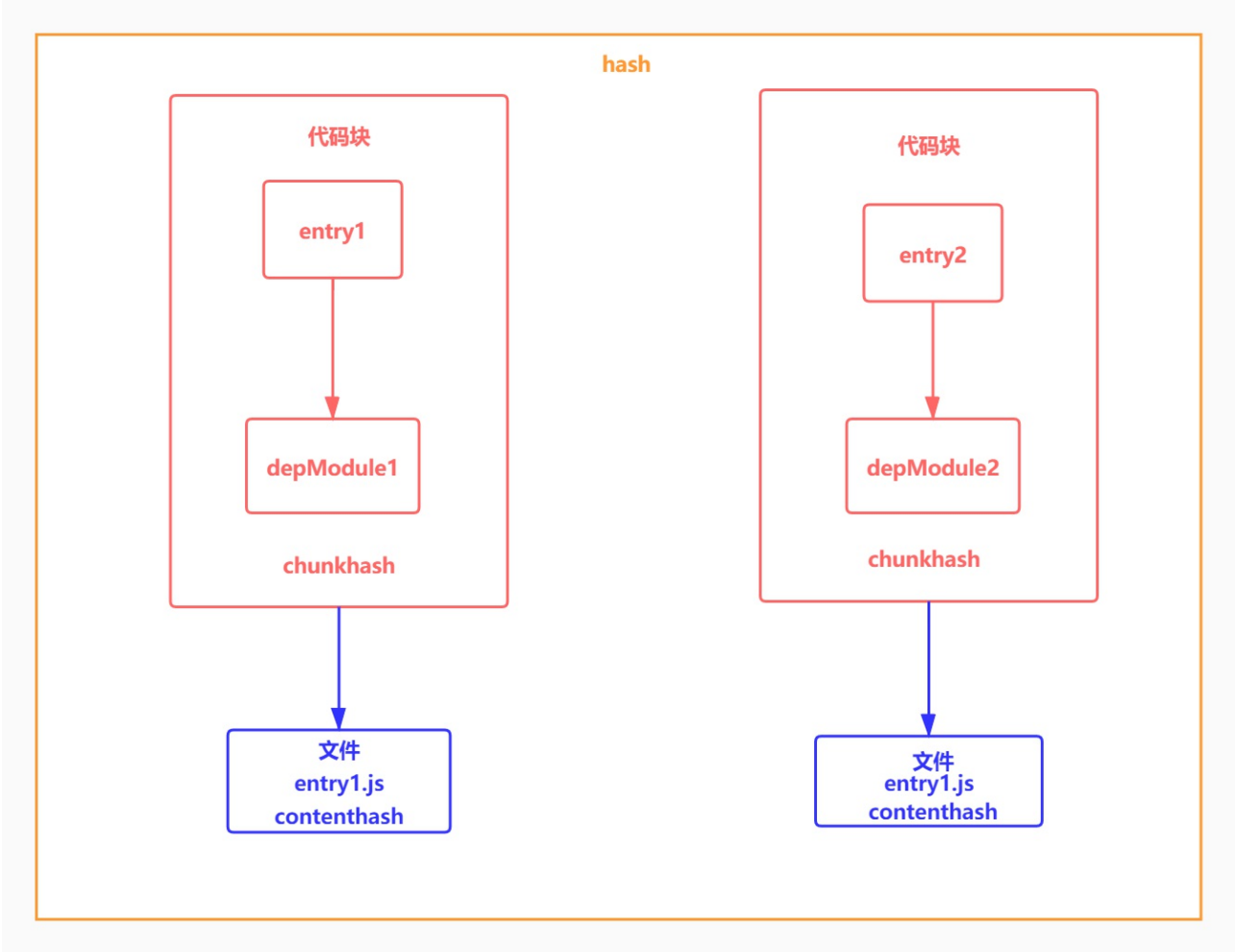
```
const path = require("path");
const glob = require("glob");
const PurgecssPlugin = require("purgecss-webpack-plugin");
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
const PATHS = {
  src: path.join(__dirname, 'src')
}
module.exports = {
  mode: "production",
  entry: {
    main: './src/index.js',
    vender:['lodash']
  },
  output:{
    path:path.resolve(__dirname,'dist'),
+    filename:'[name].[chunkhash].js'
  },
  devServer:{
    hot:false
  },
  module: {
    rules: [
      {
        test: /\.js/,
        include: path.resolve(__dirname, "src"),
        use: [
          {
            loader:'thread-loader',
            options:{
              workers:3
            }
          },
          {
            loader: "babel-loader",
            options: {
              presets: ["@babel/preset-env", "@babel/preset-react"],
            },
          },
        ],
      },
      {
        test: /\.css$/,
        include: path.resolve(__dirname, "src"),
        exclude: /node_modules/,
        use: [
          {
            loader: MiniCssExtractPlugin.loader,
          },
          "css-loader",
        ],
      },
    ],
  },
  plugins: [
    new MiniCssExtractPlugin({
+      filename: "[name].[chunkhash].css"
    }),
    new PurgecssPlugin({
      paths: glob.sync(`${PATHS.src}/**/*`,  { nodir: true }),
    }),
  ],
};
```

#### 4.3.3 contenthash #

- 使用chunkhash存在一个问题，就是当在一个JS文件中引入CSS文件，编译后它们的hash是相同的，而且只要js文件发生改变 ， 关联的css文件hash也会改变,这个时候可以使用 `mini-css-extract-plugin` 里的 `contenthash`值，保证即使css文件所处的模块里就算其他文件内容改变，只要css文件内容不变，那么不会重复构建

```
const path = require("path");
const glob = require("glob");
const PurgecssPlugin = require("purgecss-webpack-plugin");
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
const PATHS = {
  src: path.join(__dirname, 'src')
}
module.exports = {
  mode: "production",
  entry: {
    main: './src/index.js',
    vender:['lodash']
  },
  output:{
    path:path.resolve(__dirname,'dist'),
    filename:'[name].[chunkhash].js'
  },
  devServer:{
    hot:false
  },
  module: {
    rules: [
      {
        test: /\.js/,
        include: path.resolve(__dirname, "src"),
        use: [
          {
            loader:'thread-loader',
            options:{
              workers:3
            }
          },
          {
            loader: "babel-loader",
            options: {
              presets: ["@babel/preset-env", "@babel/preset-react"],
            },
          },
        ],
      },
      {
        test: /\.css$/,
        include: path.resolve(__dirname, "src"),
        exclude: /node_modules/,
        use: [
          {
            loader: MiniCssExtractPlugin.loader,
          },
          "css-loader",
        ],
      },
    ],
  },
  plugins: [
    new MiniCssExtractPlugin({
+      filename: "[name].[contenthash].css"
    }),
    new PurgecssPlugin({
      paths: glob.sync(`${PATHS.src}/**/*`,  { nodir: true }),
    }),
  ],
};
```

#### 4.3.4 hash #




```
function createHash(){
   return  require('crypto').createHash('md5');
}
let entry = {
    entry1:'entry1',
    entry2:'entry2'
}
let entry1 = 'require depModule1';
let entry2 = 'require depModule2';

let depModule1 = 'depModule1';
let depModule2 = 'depModule2';

let hash =  createHash()
.update(entry1)
.update(entry2)
.update(depModule1)
.update(depModule2)
.digest('hex');
console.log('hash',hash)

let entry1ChunkHash = createHash()
.update(entry1)
.update(depModule1).digest('hex');;
console.log('entry1ChunkHash',entry1ChunkHash);

let entry2ChunkHash = createHash()
.update(entry2)
.update(depModule2).digest('hex');;
console.log('entry2ChunkHash',entry2ChunkHash);

let entry1File = entry1+depModule1;
let entry1ContentHash = createHash()
.update(entry1File).digest('hex');;
console.log('entry1ContentHash',entry1ContentHash);

let entry2File = entry2+depModule2;
let entry2ContentHash = createHash()
.update(entry2File).digest('hex');;
console.log('entry2ContentHash',entry2ContentHash);
```

#### 4.4.HashPlugin #

- 可以自己修改各种hash值

```
class HashPlugin{
    constructor(options){
        this.options = options;
    }
    apply(compiler){
        compiler.hooks.compilation.tap('HashPlugin',(compilation,params)=>{

            compilation.hooks.afterHash.tap('HashPlugin',()=>{
                let fullhash = 'fullhash';
                console.log('本次编译的compilation.hash',compilation.hash);
                compilation.hash= fullhash;
                for(let chunk of compilation.chunks){
                    console.log('chunk.hash',chunk.hash);
                    chunk.renderedHash = 'chunkHash';
                    console.log('chunk.contentHash',chunk.contentHash);
                    chunk.contentHash= { javascript: 'javascriptContentHash','css/mini-extract':'cssContentHash' }
                }
            });
        });
    }
}
module.exports = HashPlugin;
```

webpack.config.js

```
const path = require('path');
const DonePlugin = require('./plugins/DonePlugin');
const AssetPlugin = require('./plugins/AssetPlugin');
const ZipPlugin = require('./plugins/ZipPlugin');
const HashPlugin = require('./plugins/HashPlugin');
const AutoExternalPlugin = require('./plugins/AutoExternalPlugin');
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
const HtmlWebpackPlugin = require('html-webpack-plugin');
module.exports = {
    module: {
        rules: [
            {
                test: /\.css$/,
                use: [
+                   MiniCssExtractPlugin.loader,
                    'css-loader'
                ]
            }
        ]
    },
    plugins: [
+       new HashPlugin(),
    ]
}
```

## 5.moduleIds & chunkIds的优化 #

### 5.1 概念和选项 #

- module: 每一个文件其实都可以看成一个 module
- chunk: webpack打包最终生成的代码块，代码块会生成文件，一个文件对应一个chunk
- 在webpack5之前，没有从entry打包的chunk文件，都会以1、2、3...的文件命名方式输出,删除某些些文件可能会导致缓存失效
- 在生产模式下，默认启用这些功能chunkIds: "deterministic", moduleIds: "deterministic"，此算法采用 确定性的方式将短数字 ID(3 或 4 个字符)短hash值分配给 modules 和 chunks
- chunkId设置为deterministic，则output中chunkFilename里的[name]会被替换成确定性短数字ID
- 虽然chunkId不变(不管值是deterministic | natural | named)，但更改chunk内容，chunkhash还是会改变的

可选值 含义 示例 natural 按使用顺序的数字ID 1 named 方便调试的高可读性id src_two_js.js deterministic 根据模块名称生成简短的hash值 915 size 根据模块大小生成的数字id 0

### 6.2 webpack.config.js #

webpack.config.js

```
const path = require('path');
module.exports = {
    mode: 'development',
    devtool:false,
+   optimization:{
+       moduleIds:'deterministic',
+       chunkIds:'deterministic'
+   }
}
```

### 5.3 src\index.js #

src\index.js

```
import('./one');
import('./two');
import('./three');
```