---

link: null
title: 珠峰架构师成长计划
description: src\index.js
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=124 sentences=187, words=959

---

1. webpack5新特性介绍

- 持久化缓存
- 资源模块
- moduleIds & chunkIds的优化
- 更智能的 tree shaking
- nodeJs的 polyfill脚本被移除
- 支持生成 e6/es2015的代码
- SplitChunk和模块大小
- Module Federation

2.持久化缓存

- 缓存 (https://webpack.docschina.org/configuration/other-options/#cache)生成的webpack模块和chunk,来改善构建速度
- cache 会在开发模式被设置成 type: 'memory' 而且在 生产 模式 中被禁用
- 在webpack5中默认开启，缓存默认是在内存里,但可以对 cache进行设置
- 当设置 cache.type: "filesystem"的时候,webpack会在内部启用文件缓存和内存缓存，写入的时候会同时写入内存和文件，读取缓存的时候会先读内存，如果内存里没有才会读取文件
- 每个缓存最大资源占用不超过500MB,当逼近或超过500MB时，会优先删除最老的缓存，并且缓存的有效期最长为2周
- FileMiddleware.js (node_modules_webpack@5.10.1@webpack\lib\serialization\FileMiddleware.js)
- PackFileCacheStrategy.js:1036 (node_modules_webpack@5.10.1@webpack\lib\cache\PackFileCacheStrategy.js)
- FileSystemInfo.js:1691 (node_modules_webpack@5.10.3@webpack\lib\FileSystemInfo.js)
- 默认情况下，webpack 假定 webpack 所在的 node_modules 目录只被包管理器修改。对 node_modules 来说，哈希值和时间戳会被跳过

2.1 安装

```
cnpm i webpack webpack-cli webpack-dev-server babel-loader @babel/core  @babel/preset-env -D
```

2.2 webpack.config.js

```
const path = require('path');
module.exports = {
    mode: 'development',
    cache: {
        type: 'filesystem',
        cacheDirectory: path.resolve(__dirname, 'node_modules/.cache/webpack'),
    },
    watch: true,
    module: {
        rules: [
            {
                test: /\.js$/,
                use: [
                    {
                        loader: 'babel-loader',
                        options: {
                            presets: [
                                "@babel/preset-env"
                            ]
                        }
                    }
                ]
            }
        ]
    }
}
```

2.3 package.json

```
  "scripts": {
    "build": "webpack",
    "debug": "webpack"
  },
```

3.资源模块

- 资源模块(asset module)是一种模块类型，它允许使用资源文件（字体，图标等）而无需配置额外 loader
- 在 webpack 5 之前，通常使用：
    - raw-loader 将文件导入为字符串
    - url-loader 将文件作为 data URI 内联到 bundle 中
    - file-loader 将文件发送到输出目录
- 资源模块类型(asset module type)，通过添加 4 种新的模块类型，来替换所有这些 loader
    - asset/resource 发送一个单独的文件并导出 URL。之前通过使用 file-loader 实现。
    - asset/inline 导出一个资源的 data URI。之前通过使用 url-loader 实现。
    - asset/source 导出资源的源代码。之前通过使用 raw-loader 实现。
    - asset 在导出一个 data URI 和发送一个单独的文件之间自动选择。之前通过使用 url-loader，并且配置资源体积限制实现

```
module.exports = {
  module: {
    rules: [
      {
        test: /\.png$/,
        type: 'asset/resource'
      },
      {
        test: /\.ico$/,
        type: 'asset/inline'
      },
      {
        test: /\.txt$/,
        type: 'asset/source'
      }
    ]
  },
  experiments: {
    asset: true
  },
};
```

3.1 老方式

3.1.1 src\index.js

src\index.js

```
import url from './images/kf.jpg';
let img = new Image();
img.src = url;
document.body.appendChild(img);
```

3.1.2 webpack.config.js

webpack.config.js

```
module: {
        rules: [
            {
                test: /\.(jpg|png|gif)$/,
                type:'asset'
            }
        ]
}
```

3.2 新方式

- 新的方式语法是为了允许在没有打包工具的情况下运行代码。这种语法也可以在浏览器中的原生 ECMAScript 模块中使用

3.2.1 src\index.js

src\index.js

```
let url = new URL('./images/kf.jpg', import.meta.url);
let img = new Image();
img.src = url;
document.body.appendChild(img);
```

4.URIs

- Webpack 5 支持在请求中处理协议
- 支持data 支持 Base64 或原始编码,MimeType可以在 `module.rule`中被映射到加载器和模块类型
- 支持http(s)

src\index.js

```
import data from "data:text/javascript,export default 'title'";
import url from 'https://img.zhufengpeixun.com/zfjg.png';
console.log(data,url);
```

webpack.config.js

```
const webpack = require('webpack');
const HtmlWebpackPlugin = require('html-webpack-plugin');
module.exports = {
    target: ['es6'],
    plugins:[
        new webpack.experiments.schemes.HttpsUriPlugin()
    ]
}
```

5.moduleIds & chunkIds的优化

5.1 概念

- 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

5.2 优化

- 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

可选值 含义 示例 false 不应使用任何内置算法,插件提供自定义算法 Path variable [name] not implemented in this context: [name].js 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

```
const path = require('path');
module.exports = {
    mode: 'development',
    devtool:false,
+    optimization:{
+        moduleIds:'deterministic',
+        chunkIds:'size'
+    },
    module: {
        rules: [
            {
                test: /\.js$/,
                use: [
                    {
                        loader: 'babel-loader',
                        options: {
                            presets: [
                                "@babel/preset-env"
                            ]
                        }
                    }
                ]
            }
        ]
    }
}
```

6.移除Node.js的polyfill

- webpack4带了许多Node.js核心模块的 polyfill,一旦模块中使用了任何核心模块(如crypto)，这些模块就会被自动启用
- webpack5不再自动引入这些 polyfill

6.1 安装

```
cnpm i crypto-js crypto-browserify stream-browserify buffer -D
```

6.2 src\index.js

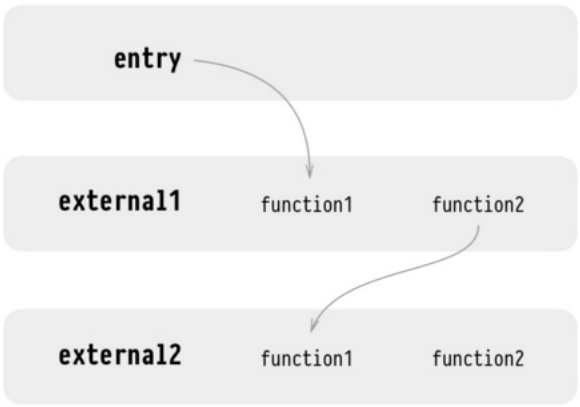
```
import CryptoJS from 'crypto-js';
console.log(CryptoJS.MD5('zhufeng').toString());
```

6.3 webpack.config.js

```
    resolve:{

        fallback:{
            "crypto":false,
            "buffer": false,
            "stream":false
        }
    },
```

7.更强大的tree-shaking

- tree-shaking (https://webpack.js.org/guides/tree-shaking/#root)
- webpack4 本身的 tree shaking 比较简单，主要是找一个 import 进来的变量是否在这个模块内出现过，非常简单粗暴



7.1 原理

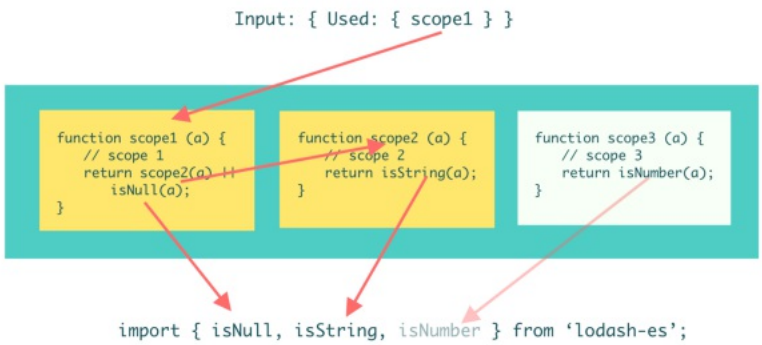
- webpack从入口遍历所有模块的形成依赖图,webpack知道那些导出被使用
- 遍历所有的作用域并将其进行分析，消除未使用的范围和模块的方法
- webpack-deep-scope-demo (https://diverse.space/webpack-deep-scope-demo/)
- webpack-deep-scope-analysis-plugin (https://github.com/vincentdchan/webpack-deep-scope-analysis-plugin)

7.1.1 作用域

- 而对于一个模块来说，只有 `class` 和 `function` 的作用域是可以导出到其他模块的

```
{
}

class Foo {

}

if (true) {

}  else {

}

for (;;) {
}

try {

} catch (e) {

}

function() {
}

switch() {
}
```

7.1.2 工作过程



7.2 开启

7.2.1 开发环境

webpack.config.js

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

module.exports = {
  entry: './src/index.js',
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist'),
  },
  mode: 'development',
   optimization: {
   usedExports: true,
  },
};
```

7.2.2 生产环境

- 生产环境默认开启

7.2.3 sideEffects

- "sideEffects": false,意思就是对所有的模块都进行 Tree Shaking
- 也就是将没有引入的方法等不进行打包到打包输出文件中

package.json

```
{"sideEffects": ["@babel/polyfill"]}
{"sideEffects": ["*.css"]}
```

7.3 嵌套的 tree-shaking

- webpack 现在能够跟踪对导出的嵌套属性的访问
- 这可以改善重新导出命名空间对象时的Tree Shaking(清除未使用的导出和混淆导出)

7.3.1 src\index.js

src\index.js

```
import * as calculator from "./calculator";
console.log(calculator.operators.add);
```

7.3.2 src\calculator.js

src\calculator.js

```
import * as operators from "./operators";
export { operators };
```

7.3.3 src\operators.js

src\operators.js

```
export const add = 'add';
export const minus = 'minus';
```

7.3.4 webpack.config.js

webpack.config.js

```
module.exports = {
    mode: 'production'
}
```

7.4 内部模块 tree-shaking

- webpack 4 没有分析模块的导出和引用之间的依赖关系
- webpack 5 可以对模块中的标志进行分析，找出导出和引用之间的依赖关系

7.4.1 src\index.js

src\index.js

```
import { getPostUrl } from './api';
console.log('getPostUrl',getPostUrl);
```

7.4.2 src\api.js

src\api.js

```
import { host } from './constants';

function useHost() {
  return host;
}

export function getUserUrl() {
  return useHost()+'/user';
}
export function getPostUrl() {
    return '/post';
}
```

7.4.3 src\api.js

src\api.js

```
export const host = 'http://localhost';
```

7.5 CommonJs Tree Shaking

- webpack 曾经不进行对 CommonJS 导出和 require() 调用时的导出使用分析
- webpack 5 增加了对一些 CommonJS 构造的支持，允许消除未使用的 CommonJs 导出，并从 require() 调用中跟踪引用的导出名称 支持以下构造：
- exports|this|module.exports.xxx = ...
- exports|this|module.exports = require("...") (reexport)

- exports|this|module.exports.xxx = require("...").xxx (reexport)
- Object.defineProperty(exports|this|module.exports, "xxx", ...)
- require("abc").xxx
- require("abc").xxx()

7.5.1 src\index.js

src\index.js

```
let api = require('./api');
console.log(api.getPostUrl);
```

7.5.2 src\api.js

src\api.js

```
function getUserUrl() {
  return '/user';
}
function getPostUrl() {
    return '/post';
}

exports.getPostUrl=getPostUrl;
```

8.splitChunks

- split-chunks-plugin (https://webpack.js.org/plugins/split-chunks-plugin/#optimizationsplitchunks)

参考

- changelog-v5 (https://github.com/webpack/changelog-v5/blob/master/README.md)