极客时间

目 录

CONTENTS

极客时间

扫码试看/订阅
《玩转webpack》

初级分析：使用 webpack 内置的 stats

stats: 构建的统计信息

package.json 中使用 stats

```
"scripts": {

  "build:stats": "webpack --env production --json > stats.json",

  ...
},
```

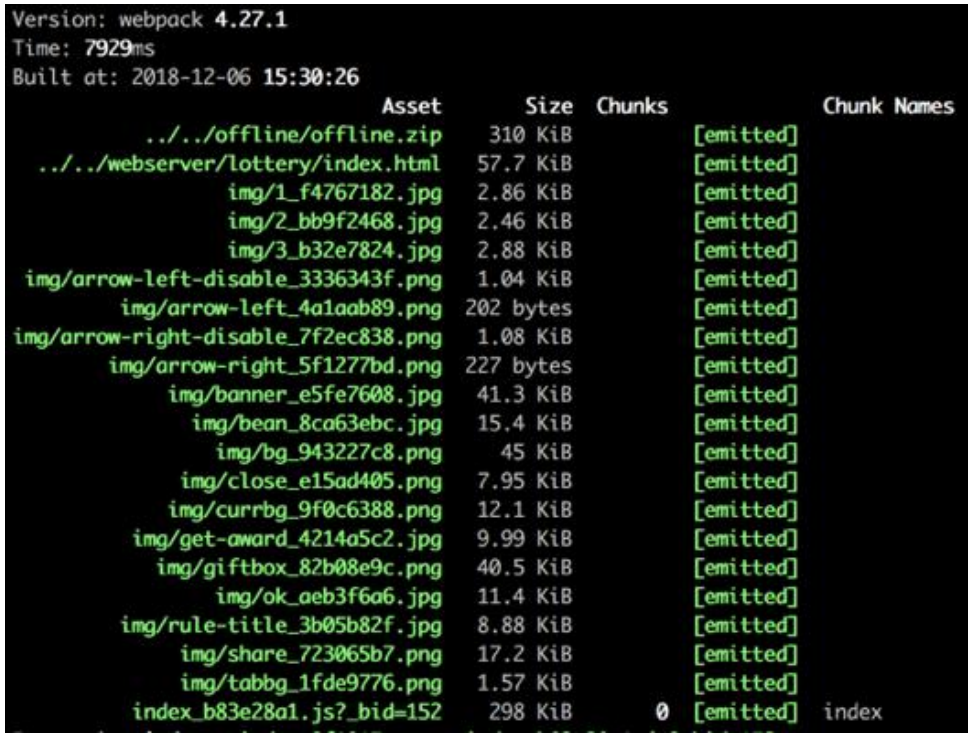
Node.js 中使用

```
const webpack = require("webpack");
const config = require("./webpack.config.js")("production");

webpack(config, (err, stats) => {
  if (err) {
    return console.error(err);
  }

  if (stats.hasErrors()) {
    return console.error(stats.toString("errors-only"));
  }

  console.log(stats);
});
```



颗粒度太粗，看不出问题所在

速度分析：使用 speed-measure-webpack-plugin

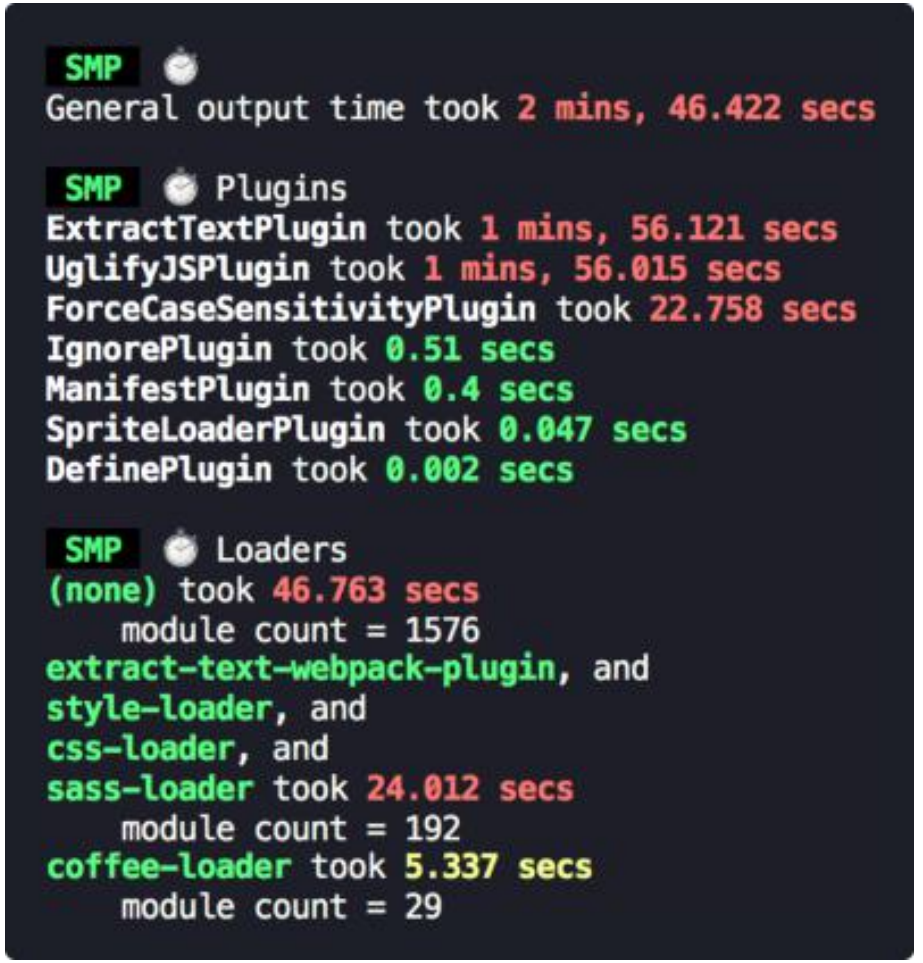
代码示例

```
const SpeedMeasurePlugin = require("speed-measure-webpack-plugin");

const smp = new SpeedMeasurePlugin();

const webpackConfig = smp.wrap({
  plugins: [
    new MyPlugin(),
    new MyOtherPlugin()
  ]
});
```

可以看到每个 loader 和插件执行耗时



速度分析插件作用

分析整个打包总耗时

每个插件和loader的耗时情况

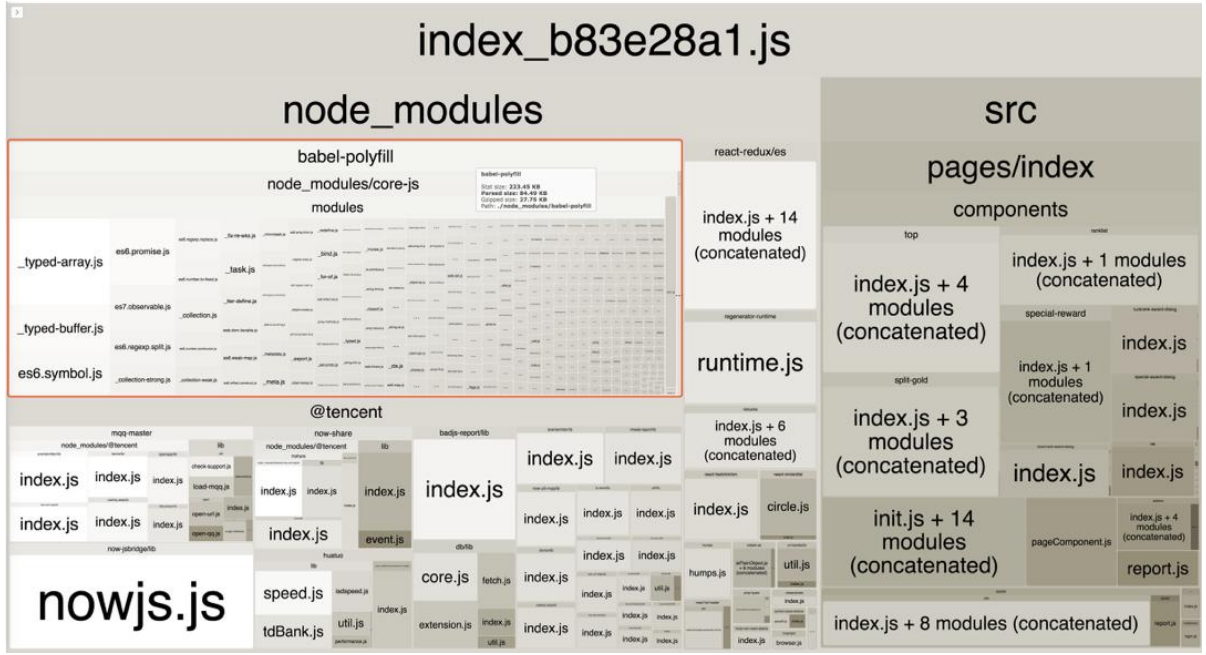
webpack-bundle-analyzer 分析体积

代码示例

```
const BundleAnalyzerPlugin = require('webpack-bundle-analyzer').BundleAnalyzerPlugin;

module.exports = {
  plugins: [
    new BundleAnalyzerPlugin()
  ]
}
```

构建完成后会在 8888 端口展示大小



可以分析哪些问题?

依赖的第三方模块文件大小

业务里面的组件代码大小

极客时间

使用高版本的 webpack 和 Node.js

```
Hash: 24c28e646ae00eae8288
Version: webpack 3.10.0
Time: 54263ms
```

```
Hash: 7bf750defb49e9a247f3
Version: webpack 4.0.0-beta.2
Time: 26563ms
Built at: 2018-2-18 02:31:50
```

构建时间降低了 60%–98%！

使用 webpack4：优化原因

V8 带来的优化（for of 替代 forEach、Map 和 Set 替代 Object、includes 替代 indexOf）

默认使用更快的 md4 hash 算法

webpacks AST 可以直接从 loader 传递给 AST，减少解析时间

使用字符串方法替代正则表达式

```
1 - crc32b 0.111036300659
2 - crc32 0.112048864365
3 - md4 0.120795726776
4 - md5 0.138875722885
5 - sha1 0.146368741989
6 - adler32 0.15501332283
7 - tiger192,3 0.177447080612
8 - tiger160,3 0.179498195648
9 - tiger128,3 0.184012889862
10 - ripemd128 0.184052705765
11 - ripemd256 0.185411214828
12 - salsa20 0.198500156403
13 - salsa10 0.204956293106
14 - haval160,3 0.206098556519
15 - haval256,3 0.206891775131
16 - haval224,3 0.206954240799
```

多进程/多实例构建：资源并行解析可选方案

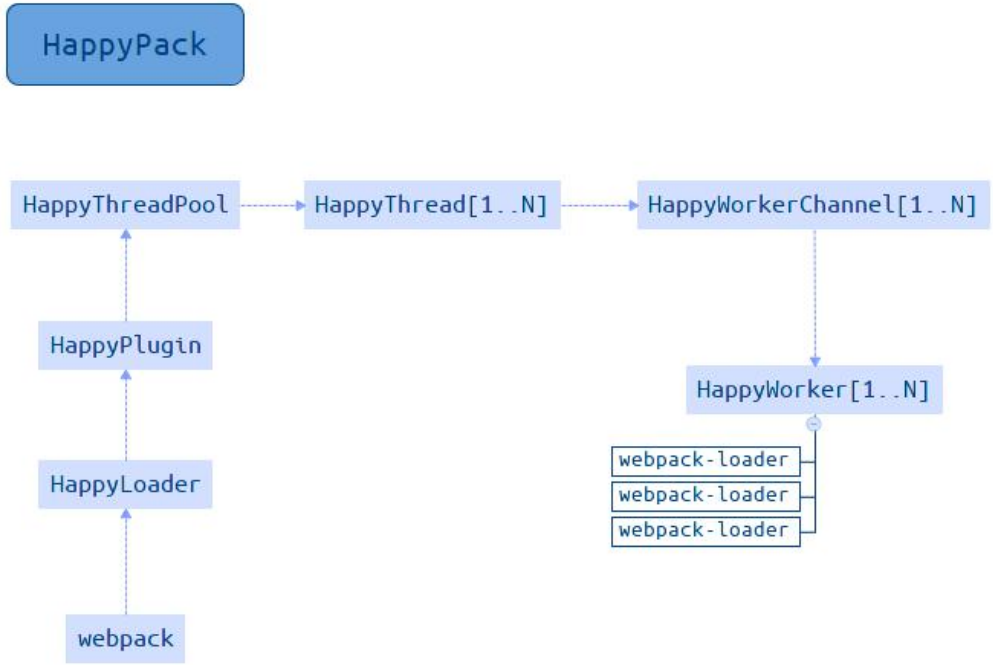
多进程/多实例：使用 HappyPack 解析资源

原理：每次 webapck 解析一个模块，HappyPack 会将它及它的依赖分配给 worker 线程中

代码示例

```
exports.plugins = [
  new HappyPack({
    id: 'jsx',
    threads: 4,
    loaders: [ 'babel-loader' ]
  }),

  new HappyPack({
    id: 'styles',
    threads: 2,
    loaders: [ 'style-loader', 'css-loader', 'less-loader' ]
  })
];
```



HappyPack工作流程

多进程/多实例：使用 thread-loader 解析资源

极客时间

原理：每次 webpack 解析一个模块，thread-loader 会将它及它的依赖分配给 worker 线程中

```
module.exports = smp.wrap({
    entry: entry,
    output: {
        path: path.join(__dirname, 'dist'),
        filename: '[name]_[chunkhash:8].js'
    },
    mode: 'production',
    module: {
        rules: [
            {
                test: /.js$/,
                use: [
                    {
                        loader: 'thread-loader',
                        options: {
                            workers: 3
                        }
                    },
                    'babel-loader',
                    // 'eslint-loader'
                ]
            },
        ]
    }
}
```

多进程/多实例：并行压缩

方法一：使用 parallel-uglify-plugin 插件

```
const ParallelUglifyPlugin = require('webpack-parallel-uglify-plugin');

module.exports = {
  plugins: [
    new ParallelUglifyPlugin({
      uglifyJS: {
        output: {
          beautify: false,
          comments: false,
        },
        compress: {
          warnings: false,
          drop_console: true,
          collapse_vars: true,
          reduce_vars: true,
        }
      },
    }),
  ],
};
```

多进程/多实例：并行压缩

方法二：uglifyjs-webpack-plugin 开启 parallel 参数

```
const UglifyJsPlugin = require('uglifyjs-webpack-plugin');

module.exports = {
  plugins: [
    new UglifyJsPlugin({
        uglifyOptions: {
            warnings: false,
            parse: {},
            compress: {},
            mangle: true,
            output: null,
            toplevel: false,
            nameCache: null,
            ie8: false,
            keep_fnames: false
        },
        parallel: true
    })
  ],
};
```

多进程/多实例：并行压缩

方法三：terser-webpack-plugin 开启 parallel 参数

```
const TerserPlugin = require('terser-webpack-plugin');

module.exports = {
  optimization: {
    minimizer: [
      new TerserPlugin({
        parallel: 4
      })
    ],
  },
};
```

分包：设置 Externals

思路：将 react、react-dom 基础包通过 cdn 引入，不打入 bundle 中

方法：使用 html-webpack-externals-plugin

```
const HtmlWebpackExternalsPlugin = require('html-webpack-externals-plugin');

plugins: [
    new HtmlWebpackExternalsPlugin({
        externals: [
            {
                module: 'react',
                entry: '//11.url.cn/now/lib/15.1.0/react-with-addons.min.js?_bid=3123',
                global: 'React'
            }, {
                module: 'react-dom',
                entry: '//11.url.cn/now/lib/15.1.0/react-dom.min.js?_bid=3123',
                global: 'ReactDOM'
            }
        ]
    }),
];
```

```
<!doctype html>
<html lang="zh_CN" style="font-size: 146.5px;">
▶<head>…</head>
▼<body style="font-size: 18px;">
  ▶<script>…</script>
  ▶<div id="container">…</div>
    <script type="text/javascript" src="//11.url.cn/now/lib/16.2.0/react.min.js?_bid=3123"></script>
    <script type="text/javascript" src="//11.url.cn/now/lib/16.2.0/react-dom.min.js?_bid=3123"></script>
    <script type="text/javascript" src="//s.url.cn/qqun/qun/qunpay/qq/withdraw/income_455d05c8.js?_bid=152"
    TWIaa8ZD/rQZptX8Urp5O2Ef3IT48JbtHS07nW2U= sha384-6E2BbRVmJ2ZLQCQyWHyOYRftEVktwLsaWnC+8h1oyip/0F+6Xa3Lo+
    "anonymous"></script>
  </body>
</html>
```

进一步分包：预编译资源模块

极客时间

思路：将 react、react-dom、redux、react-redux
基础包和业务基础包打包成一个文件

方法：使用 DLLPlugin 进行分包，DllReferencePlugin
对 manifest.json 引用

使用 DLLPlugin 进行分包

极客时间

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

module.exports = {
    context: process.cwd(),
    resolve: {
        extensions: ['.js', '.jsx', '.json', '.less', '.css'],
        modules: [__dirname, 'node_modules']
    },
    entry: {
        library: [
            'react',
            'react-dom',
            'redux',
            'react-redux'
        ]
    },
    output: {
        filename: '[name].dll.js',
        path: path.resolve(__dirname, './build/library'),
        library: '[name]'
    },
    plugins: [
        new webpack.DllPlugin({
            name: '[name]',
            path: './build/library/[name].json'
        })
    ]
};
```

使用 DllReferencePlugin 引用 manifest.json

极客时间

在 webpack.config.js 引入

```
module.exports = {
  plugins: [
    new webpack.DllReferencePlugin({
      manifest: require('./build/library/manifest.json')
    })
  ]
};
```

引用效果

```
    </div>
  </div>
  <script src="/build/library/library.dll.js"></script>
  <script type="text/javascript" src="//s.url.cn/qqun/qun/qunpay/qq/withdraw/income_455d05c8.js?_bid=152"
  integrity="sha256-6L/TWIaa8ZD/rQZptX8Urp5O2Ef3IT48JbtHS07nW2U= sha384-
  6E2BbRVmJ2ZLQCQyWHyOYRftEVktwLsaWnC+8h1oyip/0F+6Xa3Lo+ULJ7hcLTEy" crossorigin="anonymous"></script>
```

缓存

目的：提升二次构建速度

缓存思路：

- babel-loader 开启缓存
- terser-webpack-plugin 开启缓存
- 使用 cache-loader 或者 hard-source-webpack-plugin

缩小构建目标

目的：尽可能的少构建模块

比如 babel-loader 不解析 node_modules

```
module.exports = {
    rules: {
        test: /\.js$/,
        loader: 'happypack/loader',
        exclude: 'node_modules'
    }
}
```

减少文件搜索范围

优化 resolve.modules 配置（减少模块搜索层级）

优化 resolve.mainFields 配置

优化 resolve.extensions 配置

合理使用 alias

```
module.exports = {
    resolve: {
        alias: {
            react: path.resolve(__dirname, './node_modules/react/dist/react.min.js'),
        },
        modules: [path.resolve(__dirname, 'node_modules')],
        extensions: ['.js'],
        mainFields: ['main'],
    }
};
```

图片压缩

要求：基于 Node 库的 imagemin 或者 tinypng API

使用：配置 image-webpack-loader

```
return {
    test: /\.(png|svg|jpg|gif|blob)$/,
    use: [{
        loader: 'file-loader',
        options: {
            name: `${filename}img/[name]${hash}.[ext]`
        }
    }, {
        loader: 'image-webpack-loader',
        options: {
            mozjpeg: {
                progressive: true,
                quality: 65
            },
            optipng: {
                enabled: false,
            },
            pngquant: {
                quality: '65-90',
                speed: 4
            },
            gifsicle: {
                interlaced: false,
            },
            webp: {
                quality: 75
            }
        }
    }]
};
```

Imagemin的优点分析

有很多定制选项

可以引入更多第三方优化插件，例如pngquant

可以处理多种图片格式

Imagemin的压缩原理

pngquant: 是一款PNG压缩器，通过将图像转换为具有alpha通道（通常比24/32位PNG文件小60–80%）的更高效的8位PNG格式，可显著减小文件大小。

pngcrush:其主要目的是通过尝试不同的压缩级别和PNG过滤方法来降低PNG IDAT数据流的大小。

optipng:其设计灵感来自于pngcrush。optipng可将图像文件重新压缩为更小尺寸，而不会丢失任何信息。

tinypng:也是将24位png文件转化为更小有索引的8位图片，同时所有非必要的metadata也会被剥离掉

tree shaking(摇树优化)复习

极客时间

概念：1 个模块可能有多个方法，只要其中的某个方法使用到了，则整个文件都会被打到 bundle 里面去，tree shaking 就是只把用到的方法打入 bundle ，没用到的方法会在 uglify 阶段被擦除掉。

使用：webpack 默认支持，在 .babelrc 里设置 modules: false 即可

- production mode的情况下默认开启

要求：必须是 ES6 的语法，CJS 的方式不支持

无用的 CSS 如何删除掉?

PurifyCSS: 遍历代码，识别已经用到的 CSS class

uncss: HTML 需要通过 jsdom 加载，所有的样式通过PostCSS解析，通过 document.querySelector 来识别在 html 文件里面不存在的选择器

在 webpack 中如何使用 PurifyCSS?

使用 purgecss-webpack-plugin

- https://github.com/FullHuman/purgecss-webpack-plugin

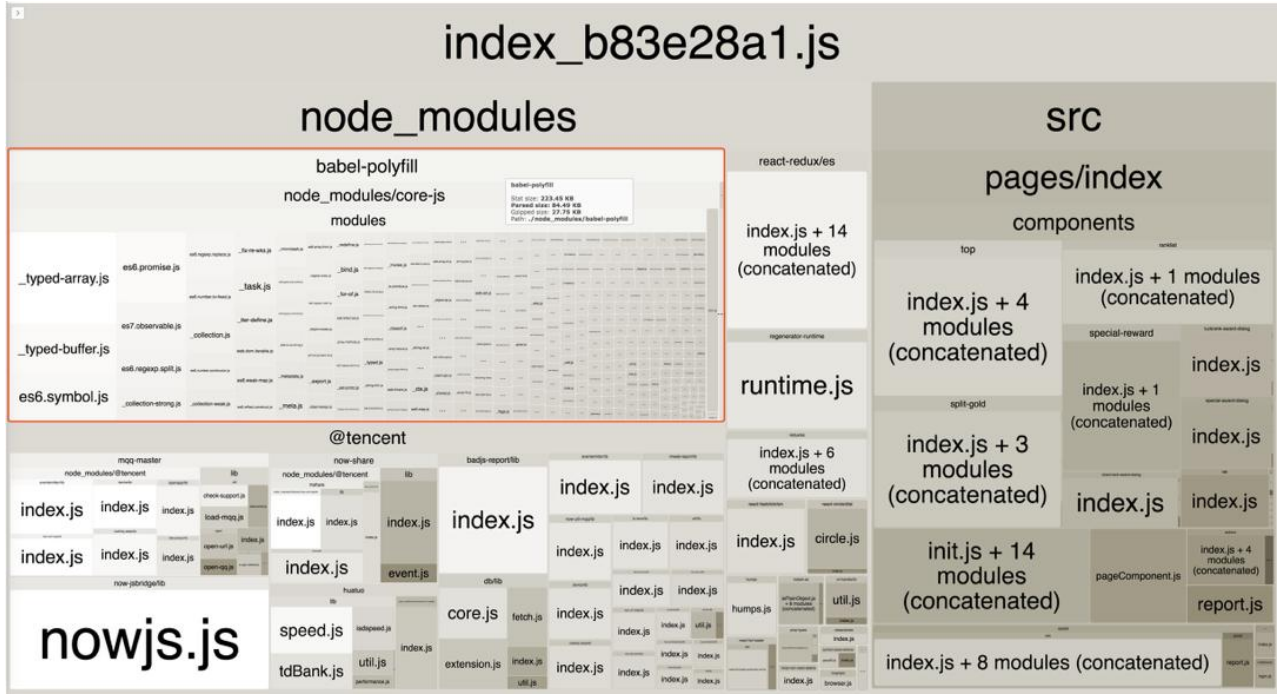
和 mini-css-extract-plugin 配合使用

```
const path = require('path')
const glob = require('glob')
const MiniCssExtractPlugin = require('mini-css-extract-plugin')
const PurgecssPlugin = require('purgecss-webpack-plugin')

const PATHS = {
  src: path.join(__dirname, 'src')
}

module.exports = {
  module: {
    rules: [
      {
        test: /\.css$/,
        use: [
          MiniCssExtractPlugin.loader,
          "css-loader"
        ]
      }
    ]
  },
  plugins: [
    new MiniCssExtractPlugin({
      filename: "[name].css",
    }),
    new PurgecssPlugin({
      paths: glob.sync(`${PATHS.src}/**/*`,  { nodir: true }),
    }),
  ]
}
```

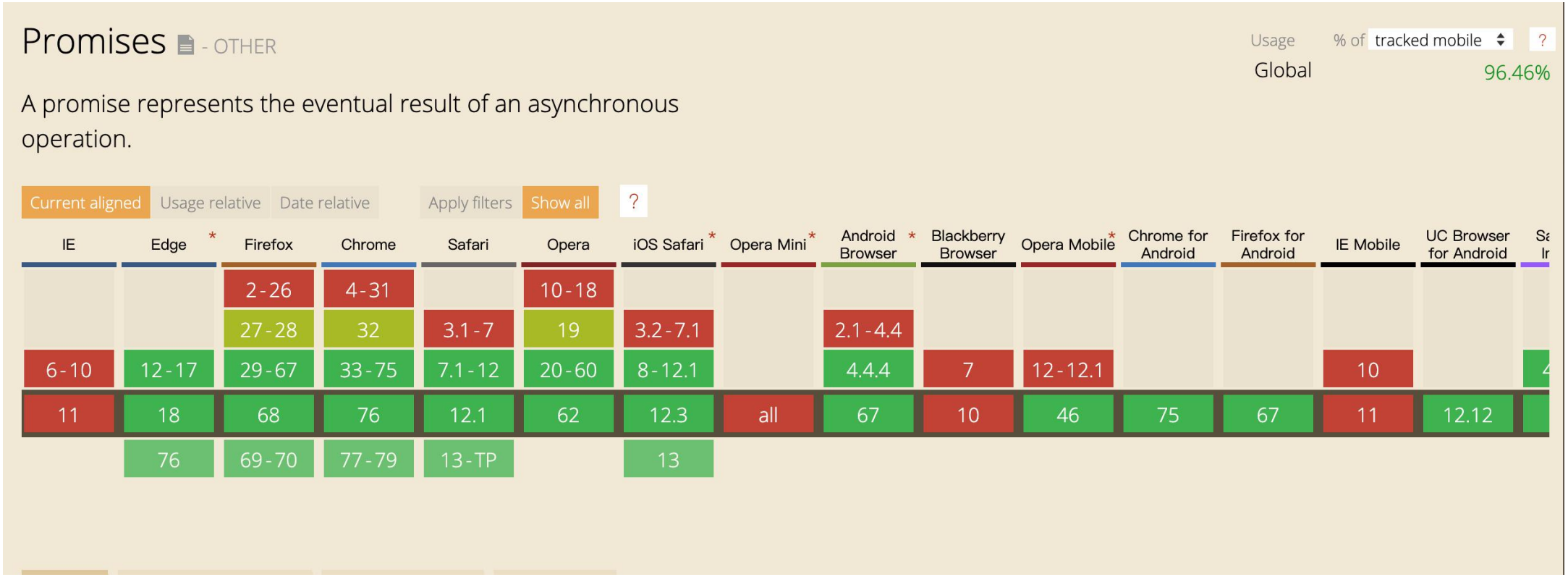
构建体积优化：动态 Polyfill



babel-polyfill

打包后体积：88.49k，占比 29.6%

Promise 的浏览器支持情况



Promises - OTHER

A promise represents the eventual result of an asynchronous operation.

Usage: % of tracked mobile — Global 96.46%

Current aligned | Usage relative | Date relative | Apply filters | Show all | ?

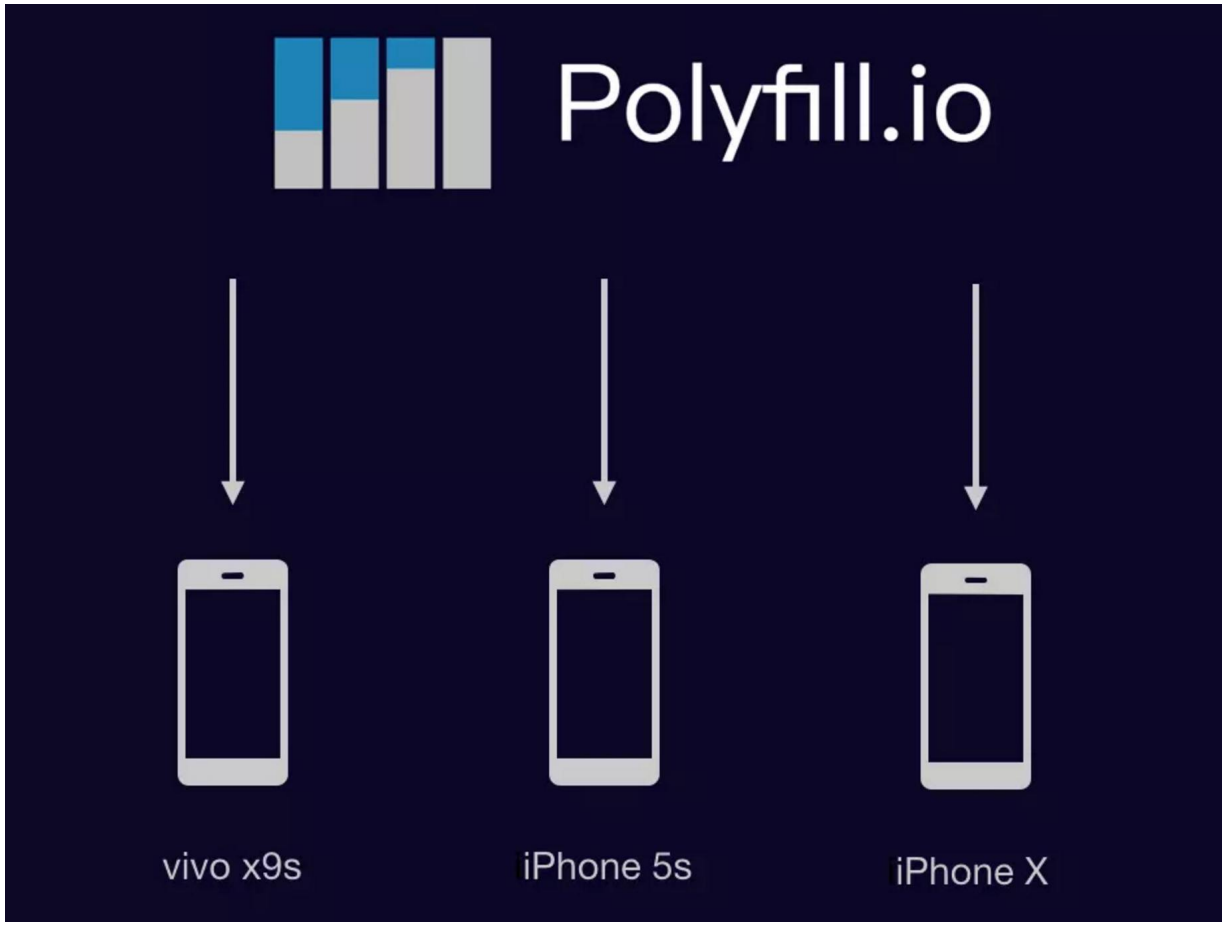
IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Blackberry Browser	Opera Mobile *	Chrome for Android	Firefox for Android	IE Mobile	UC Browser for Android	Sa Ir
		2-26	4-31		10-18										
		27-28	32	3.1-7	19	3.2-7.1		2.1-4.4							
6-10	12-17	29-67	33-75	7.1-12	20-60	8-12.1		4.4.4	7	12-12.1			10		4
11	18	68	76	12.1	62	12.3	all	67	10	46	75	67	11	12.12	
	76	69-70	77-79	13-TP		13									

构建体积优化：动态 Polyfill

方案	优点	缺点	是否采用
babel-polyfill	React官方推荐	1、包体积200K+，难以单独抽离Map、Set 2、项目里react是单独引用的cdn，如果要用它需要单独构建一份放react前加载	❌
babel-plugin-transform-runtime	能只polyfill用到类或方法，对体积较小	不能polyfill原型上的方法，不适用于业务项目的杂开发环境	❌
自己写Map、Set的polyfill	定制化高，体积小	1、重复造轮子，容易在日后踩坑成为坑 2、即使体积小，然所有用户都加载	❌
polyfill-service	只给用户返回需要的polyfill，社区维护	部分国内奇葩浏览器UA可能无法识别（但可以降级返回所需全部polyfill）	✅

Polyfill Service原理

识别 User Agent，下发不同的 Polyfill



构建体积优化：如何使用动态 Polyfill service

极客时间

polyfill.io 官方提供的服务

```
<script src="https://cdn.polyfill.io/v2/polyfill.min.js"></script>
```

基于官方自建 polyfill 服务

```
//huayang.qq.com/polyfill_service/v2/polyfill.min.js?unknown=polyfill&features=Promise,Map,Set
```

体积优化策略总结

Scope Hoisting

Tree-shaking

公共资源分离

图片压缩

动态 Polyfill

极客时间

扫码试看/订阅
《玩转webpack》