



# webpack

1

**Что такое webpack?**

# Concepts

• • •

At its core, webpack is a static module bundler for modern JavaScript applications. When webpack processes your application, it recursively builds a dependency graph that includes every module your application needs, then packages all of those modules into one or more bundles.

1 1

**Сравнение с grunt и  
gulp**

GRUNT



**1 697 281** — downloads in the last month (npm);

**11 713** — stars on github;

GULP



**2 932 030** — downloads in the last month (npm);

**28 596** — stars on github;

WEBPACK



**9 411 991** — downloads in the last month (npm);

**36 463** — stars on github;

npm



1.2

# Принцип работы

# Что нам говорит webpack?

•••

- Все что есть на проекте может быть модулем. Будь то картинка, риг-файл или файл стилей могут подключаться в js-файл.
- За работу с разными типами модулей отвечает файл конфигурации.
- Функционал webpack может расширяться лоадерами и плагинами.

```
1 // webpack.config.js
2 module.exports = {
3   entry: './main.js',
4   output: {
5     filename: 'bundle.js'
6   }
7 };
```

```
1 // index.html
2 <html>
3   <body>
4     <script type="text/javascript" src="bundle.js"></script>
5   </body>
6 </html>
7
```

2

**Entry**

**Entry point** указывает webpack`у с какого файла нужно начинать строить зависимости.

Точек входа может быть как одна так и несколько. Если несколько, то они указываются в объекте парой «ключ-значение» (где ключ — это имя точки, а значение — путь к файлу)

# Одна точка входа

```
1 // webpack.config.js
2 module.exports = {
3   entry: './main.js',
4   output: {
5     filename: 'bundle.js'
6   }
7 };
```

# Несколько точек входа

```
1 module.exports = {  
2   entry: {  
3     bundle1: './main1.js',  
4     bundle2: './main2.js'  
5   },  
6   output: {  
7     filename: '[name].js'  
8   }  
9 };
```

3

# Output

**OutPut показывает куда  
будут выгружаться  
файлы, как они будут  
называться и как будут  
меняться ссылки.**

# Пример

```
1 const path = require('path');
2
3 module.exports = {
4   entry: {
5     bundle1: './main1.js',
6     bundle2: './main2.js'
7   },
8   output: {
9     publicPath: '/',
10    path: path.resolve(__dirname, 'dist/js'),
11    filename: '[name].[hash].js'
12  }
13 }
```

4

# Модули

# Какие бывают модули?

• • •

- **AMD** — одна из самых древних систем организации модулей;
- **CommonJS** — система модулей, встроенная в сервер Node.js.
- **UMD** — система модулей, которая предложена в качестве универсальной.
- **ES6 Modules** — модули принятые стандартом

# CommonJS

```
1 //module.js
2 module.exports = function () {
3     alert('hello');
4 }
5
```

```
1 //main.js
2 const sayHello = require('./module.js');
3
4 sayHello();
5
```

# ES6

```
1 //module.js
2 export default function () {
3     alert('hello');
4 }
5
```

```
1 //main.js
2 import sayHello from './module.js';
3
4 sayHello();
5
```

5

# Loaders

**Лоадеры работают непосредственно с файлами модулей. Они говорят вебпаку как файл будет компилироваться и обрабатываться.**

```
1 //main.js
2 require('./style.css')
3
```

```
1 /*style.css*/
2 body {
3   background: url('./image.png');
4 }
```

# css-loader + style-loader

```
1 const path = require('path');
2
3 module.exports = {
4   entry: {
5     main: './main.js'
6   },
7   output: {
8     publicPath: '/',
9     path: path.resolve(__dirname, 'dist'),
10    filename: './js/[name].js'
11  },
12  module: {
13    rules: [
14      {
15        test: /\.css$/,
16        use: [
17          'style-loader',
18          {
19            loader: "css-loader",
20            options: {
21              sourceMap: true
22            }
23          }
24        ]
25      }
26    ]
27  }
28}
```

```
11  },
12  module: {
13    rules: [
14      {
15        test: /\.css$/,
16        use: [
17          'style-loader',
18          {
19            loader: "css-loader",
20            options: {
21              sourceMap: true
22            }
23          }
24        ]
25      }
26    ]
27  }
28}
```

# file-loader

```
28      ],
29
30  },
31  {
32    test: /\.(\png|gif|jpe?g)$/i,
33    use: [
34      {
35        loader: 'file-loader',
36        options: {
37          name: 'images/[path][name].[ext]'
38        }
39      }
40    ],
41  }
42 ]
43 }
```

## ОСНОВНЫЕ ЛОАДЕРЫ:

- **babel-loader**
- **html-loader**
- **pug-html-loader**
- **css-loader**
- **less-loader**
- **sass-loader**
- **file-loader**
- **url-loader**
- **extract-loader**

# 6

# Plugins

**Плагины работают с самим бандлом.  
Они отвечают за расширения всего  
функционала.**

**Примеры функционала:**

- Удаление и создания новых файлов
- Минификация
- Сборка одного или нескольких файлов

# ExtractTextWebpackPlugin

```
1 const path = require('path');
2 const ExtractTextPlugin = require("extract-text-webpack-plugin");
```

```
13 module: {
14   rules: [
15     {
16       test: /\.css$/,
17       use: ExtractTextPlugin.extract({
18         fallback: "style-loader",
19         use: [
20           {
21             loader: "css-loader",
22             options: {
23               sourceMap: true
24             }
25           }
26         ]
27       })
28     },
29   ],
30 }
```

```
31 },
32 },
33 },
34 },
35 },
36 },
37 },
38 },
39 },
40 },
41 },
42 plugins: [
43   new ExtractTextPlugin("css/styles.css"),
44 ]
45 }
```

## ОСНОВНЫЕ ПЛАГИНЫ:

- **CommonsChunkPlugin(?)**
- **DefinePlugin**
- **ExtractTextWebpackPlugin**
- **HotModuleReplacementPlugin**
- **ProvidePlugin**
- **UglifyjsWebpackPlugin**

# Итого:

Entry

Output

Loaders

Plugins

```
1 const path = require('path');
2 const ExtractTextPlugin = require("extract-text-webpack-plugin");
3
4 module.exports = {
5   entry: {
6     main: './main.js' ←
7   },
8   output: {
9     publicPath: '/',
10    path: path.resolve(__dirname, 'dist'),
11    filename: './js/[name].js' ←
12  },
13  module: {
14    rules: [
15      {
16        test: /\.css$/,
17        use: ExtractTextPlugin.extract({
18          fallback: "style-loader",
19          use: [
20            {
21              loader: "css-loader",
22              options: {
23                sourceMap: true
24              }
25            }
26          ]
27        })
28      },
29      {
30        test: /\.(png|gif|jpe?g)$/i,
31        use: [
32          {
33            loader: 'file-loader',
34            options: {
35              name: 'images/[path][name].[ext]'
36            }
37          }
38        ]
39      }
40    ],
41  },
42  plugins: [
43    new ExtractTextPlugin("css/styles.css"),
44  ]
45}
```

# Process

• • •

Entry                  Loaders                  Plugins                  Output





7

## Настройка среды разработки

# NODE\_ENV

• • •

**Это глобальная переменная которую можно задавать в CLI и использовать в файле конфигурации.**

```
1 "scripts": {  
2   "build": "cross-env NODE_ENV=build ./node_modules/.bin/webpack",  
3   "watch": "cross-env NODE_ENV=dev ./node_modules/.bin/webpack"  
4 }  
5
```

# Пример

```
3
4 const NODE_ENV = process.env.NODE_ENV || 'dev';
5
```

```
14 },
15 watch: NODE_ENV === 'dev',
16 devtool: NODE_ENV === 'dev' ? "source-map" : false,
17 module: {
```



Я хочу услышать  
три самых главных  
слова

Webpack  
Dev  
Server

7.1

**webpack-dev-server**

```
1 "scripts": {  
2     "build": "cross-env NODE_ENV=build ./node_modules/.bin/webpack",  
3     "watch": "cross-env NODE_ENV=dev ./node_modules/.bin/webpack",  
4     "dev": "cross-env NODE_ENV=dev ./node_modules/.bin/webpack-dev-server"  
5 }  
6
```

```
16     devtool: NODE_ENV === 'dev' ? "source-map" : false,  
17     devServer: {  
18         contentBase: path.resolve(__dirname, "dist"),  
19         open: true  
20     },  
21     module: {
```

7.2

## Hot Module Replacement

# HMR

• • •

**Меняет, добавляет и убирает  
модули в приложении без полной  
перезагрузки**

```
23 devServer: {  
24   contentBase: path.resolve(__dirname, "dist"),  
25   open: true,  
26   hot: true  
27 },  
28   module: {  
29     rules: [
```

# 8

**Разделение кода  
на части (chunks)**

# Code Splitting

• • •

- **Entry Points:** разделение кода при помощи entry.
- **CommonsChunkPlugin:** плагин который ищет общие модули в entry points и выносит их в отдельный файл.
- **Асинхронные импорты:** подключение модулей асинхронно по мере их потребности.

# Entry

• • •

Разделение кода при  
помощи entry.

```
1<
13 module.exports = {
14   entry: {
15     main: './main.js',
16     entry2: './entry2.js',
17     entry3: './entry3.js'
18   },

```

# CommonsChunkPlugin

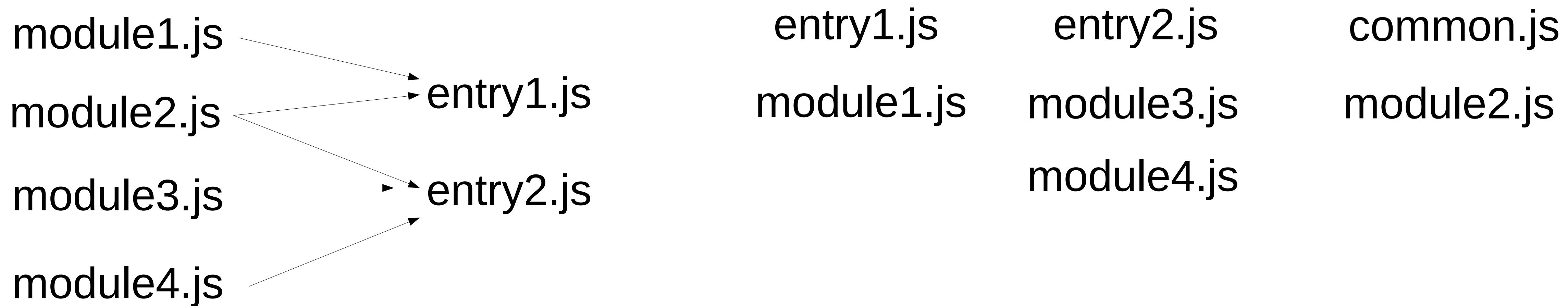
• • •

Плагин который ищет общие  
модули в entry points и выносит их в  
отдельный файл.

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

```
60   plugins: [
61     new ExtractTextPlugin("css/styles.css"),
62     new webpack.optimize.CommonsChunkPlugin({
63       name: "commons",
64       filename: "js/commons.js",
65     })
66   ]
67 }
```

# CommonsChunkPlugin



```
1 <script src="commons.js" charset="utf-8"></script>
2 <script src="entry1.js" charset="utf-8"></script>
```

# Асинхронные импорты

• • •

Require.ensure &  
import();

# require.ensure

```
1 const button = document.querySelector('.button');
2
3 button.addEventListener('click', () => {
4   require.ensure(['./button-click.js'], require => {
5     const buttonClick = require('./button-click.js')
6
7     buttonClick();
8   })
9 })
10
```

\* Создастся новый бандл который будет подключаться по востребованости

# import()

```
11
12 const button = document.querySelector('.button');
13
14 button.addEventListener('click', () => {
15   import('./button-click.js').then(buttonClick => {
16     buttonClick();
17   })
18 })
19
```

9

**Пример полной  
конфигурации для МРА**

# Что в себя включает?

...

- Сбор стилей в один файл
- Минификация
- Pug, SCSS
- ES6
- Динамическое подключение модулей

# Подключение плагинов и env

```
1 const webpack = require('webpack'),  
2     path = require('path'),  
3     UglifyJSPlugin = require('uglifyjs-webpack-plugin'),  
4     ExtractTextPlugin = require("extract-text-webpack-plugin"),  
5     CleanWebpackPlugin = require('clean-webpack-plugin'),  
6     autoprefixer = require('autoprefixer');  
7  
8 const NODE_ENV = process.env.NODE_ENV || 'dev';  
9 const DEV = NODE_ENV === 'dev';  
10
```

# Entry and output

```
11 module.exports = {
12   context: path.resolve(__dirname, 'app'),
13   entry: {
14     main: './main.js',
15     trash: './pug-and-style.js'
16   },
17   output: {
18     publicPath: '/',
19     path: path.resolve(__dirname, 'dist'),
20     filename: './js/[name].js'
21   },
22 }
```

# Devtools

```
22     watch: DEV,  
23     devtool: DEV ? "source-map" : false,  
24     watchOptions: {  
25       aggregateTimeout: 100  
26     },  
27     devServer: {  
28       contentBase: path.resolve(__dirname, "dist"),  
29       open: true  
30     },  
31     modules: [
```

# Loaders

• • •

- babel-loader
- file-loader
- url-loader
- extract-loader
- html-loader
- pug-html-loader
- css-loader
- resolve-url-loader
- postcss-loader
- sass-loader

# Plugins

```
118      ],
119    },
120    plugins: [
121      new webpack.NoEmitOnErrorsPlugin(),
122      new ExtractTextPlugin({
123        filename: "./css/style.css"
124      })
125    ]
126  };
127
128 if (NODE_ENV === 'build') {
129   module.exports.plugins.push(new UglifyJSPlugin());
130   module.exports.plugins.push(new CleanWebpackPlugin('dist'));
131 }
```

# pug-and-style.js

```
1 function requireAll(context) {
2     context.keys().forEach(context);
3 }
4
5 requireAll(require.context('./', false, /\.pug/));
6 requireAll(require.context('./', true, /\.s?css/));
7
```



9

**Что дальше?**

# Parcel

•••

Parcel —  
маленький и  
быстрый бандлер,  
позиционируется  
как решение для  
маленьких  
проектов.



# Бенчмарки

Упаковщик	Время, сек.
browserify	22,98
webpack	20,71
parcel	9,98
parcel (с кэшем)	2,64

За основу было взято приложение с достаточно большим размером, содержащее 1726 модулей, 6,5 МБ несжатого кода. Бандл был сделан на 2016 MacBook Pro с 4-ядерным ЦП.

**и т.д.**

...

- Webpack 4 уже скоро!!!
- Gulp and webpack-stream

# Thank you!

Github: <https://github.com/rotarNikita/webpack-sample>  
Telegram: @nikitosinar (<https://t.me/nikitosinar>)