УДК 004

Использования инструмента Webpack для сборки веб-приложений

Круглик Роман Игоревич
Приамурский государственный университет им. Шолом-Алейхема
Студент

Аннотация
В статье рассматриваются базовые функции для работы с Webpack. Так же приведён пример подключения css файла.
Ключевые слова: Webpack, веб-приложение, сборка приложений.

Using the Webpack tool to build web applications

Kruglik Roman Igorevich
Sholom-Aleichem Priamursky State University
Student

Abstract
In article discusses the basic functions for working with Webpack. An example of connecting a css file is also given.
Keywords: Webpack, web application, application assembly.

В последние несколько лет, в индустрии web-разработки наблюдается постоянный рост одностраничных приложений (SPA). К сожалению, как известно, чем больше функций, тем больше кода приходится писать и, учитывая среднюю сложность SPA, в конце концов достигается точка, где код может состоять из десятков, если не из сотен .js, .css и многих других файлов.

Поэтому возникает следующая проблема: как справиться с кодовой базой такого размера? Как управлять порядком загрузки файлов/модулей, делающих приложение работоспособным, когда их так много? Раньше можно было просто вставить в HTML теги script в определенном порядке. Подключение 3 js файла выглядило так (см. рис. 1).

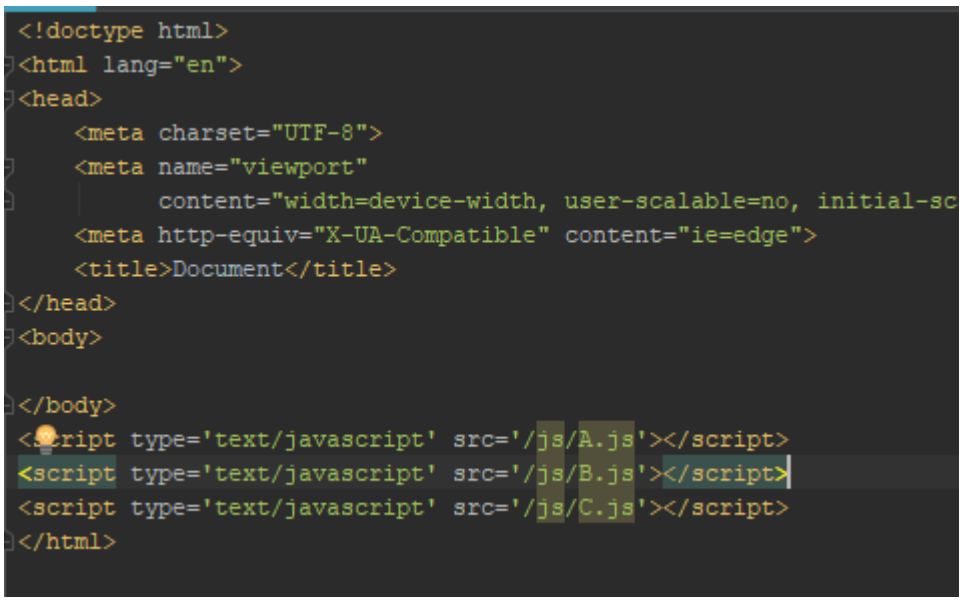

Рисунок 1. Расположение подключаемых файлов

После модификаций или исправлений количество файлов будет постепенно расти, тем самым и усложняется управление зависимостями.

Поэтому необходимо создавать единственный .js файл (или как можно меньше) из всех этих отдельно взятых A.js, B.js, C.js и так далее, чтобы избежать затрат на несколько HTTP запросов загрузки для каждого .js файла. Этот процесс называется сборка, он объединяет разные файлы в один, сохраняя порядок их зависимостей. Именно поэтому был создан Webpack.

Webpack — это инструмент, основная цель которого состоит в том, чтобы собрать все .js файлы в любое нужное количество пакетов (bundles), а также убедиться, что в собранном пакете есть все .js файлы проекта в правильном порядке.

Областью использования сборки веб-приложений через webpack интересуются многие. В статье И.В. Попков, Л.В. Курзаева [1] описывается среда для сборки приложения, при помощи Webpack. А.А. Штыков [2] проводит обзор современных стандартов в области организации труда front-end разработчика. Перфильев Н.В., Макаров Д.А. [3] предлагают рассмотреть метод оптимизации сборки frontend-приложений.

В данной статье рассматриваются базовые функции для работы с Webpack. Так же приведён пример подключения css файла.

Для начала необходимо установить данный инструмент и это можно сделать через консоль «npm install --save-dev webpack».

Webpack достаточно сложный и в тоже время гибкий инструмент, поэтому для начала необходимо изучить базовые понятия:

1. Entry - вход
2. Output - вывод
3. Loader - загрузчик
4. Plugin - плагин

Entry указывает Webpack о главном файле. Он является точкой входа, которая будет использована для построения внутреннего графа

зависимостей. Все зависимости превращаются в файлы, которые называются bundles (пакеты или узлы). Простой пример использования entry (см. рис. 2).

```
module.exports = {

    entry: './js/file.js'

};
```

Рисунок 2. Использование Entry

Следующее понятие output указывает путь до файла, который будет связываться с HTML кодом. Этот файл будет размещать в себя всю сборку собранных bundles. Простой пример использования output (см. рис. 3).

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

module.exports = {

    entry: './js/file.js',

    output: {

        path: path.resolve(__dirname, 'dist'),

        filename: 'my-first-webpack.bundle.js'

    }

};
```

Рисунок 3. Использование output

Переменная path хранит в себе путь, а filename – имя файла. Так же существуют loaders, которые помогают обрабатывать различные файлы, потому что сам Webpack понимает только JavaScript код. Загрузчики трансформируют все типы файлов в модули, которые затем можно добавить в граф зависимостей создаваемого приложения (см. рис. 4).

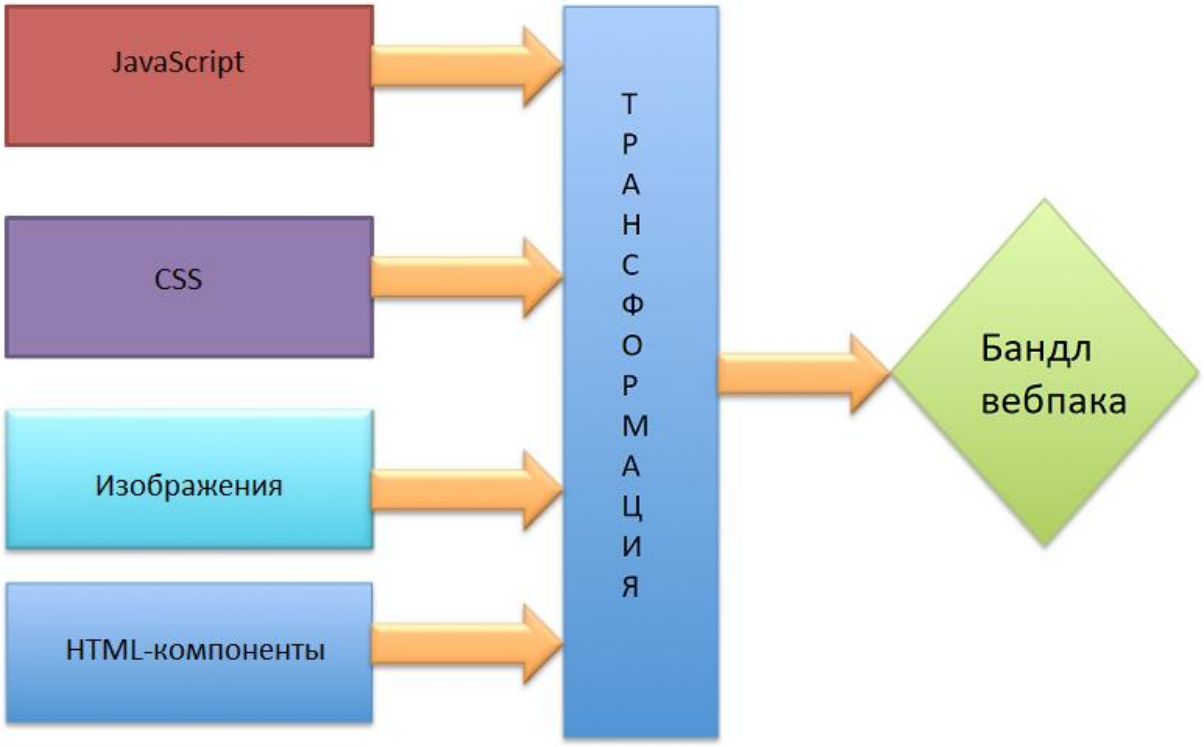

Рисунок 4. Трансформация файлов в бандл

Если загрузчики используются для трансформации определенных типов модулей, то плагины могут быть использованы для выполнения гораздо более широкого списка задач.

Для того, чтобы использовать плагин, необходимо использовать require() и добавить его в массив плагинов.

После того, как нам стало известно об основных понятиях рассмотрим пример подключения стилей через сборку. Чтобы работать со стилями, необходимо установить 2 плагина через консоль:

1. npm install style-loader –save (чтобы работать с css)
2. npm install css-loader –save (грузить и парсить css файлы)

Создадим HTML файл, к которому подключим всего один js и css файл (см. рис. 5).

```
<html>
    <head>
        <meta charset="utf-8">
        <link rel="stylesheet" type="text/css" href="bundle.css">
    </head>
    <body>
        Webpack
        <script type="text/javascript" src="bundle.js" charset="utf-8"></script>
    </body>
</html>
```

Рисунок 5. HTML файл

Далее добавим установленный загрузчик в config Webpack (см. рис. 6).

```
module.exports = {
    entry: './main.js',
    output: {
        filename: 'bundle.js'
    },
    resolve: {
        modulesDirectories: ['node_modules']
    },
    module: {
        loaders: [
            {
                test: /\.js/,
                loader: 'babel',
                exclude: /(node_modules|bower_components)/
            },
            {
                test: /\.css$/,
                loader: 'style!css'
            }
        ]
    }
};
```

Рисунок 6. Config webpack

За загрузчик отвечает параметр module, в котором:

1. test является регулярным выражением для парсинга файлов.
2. loader отвечает за путь подключаемого файла.

Теперь создадим css файл и впишем в него простой код для смены цвета body (см. рис. 7).

```
body {
    background: grey;
}
```

Рисунок 7. CSS файл

Точка входа определена как main.js. Необходимо подключить в ней стили (см. рис. 8).

```
import './main.css';
```

Рисунок 8. Точка входа main.js

Всё готово и можно посмотреть на результат. Страница должна быть окрашена в серый цвет (см. рис. 9).

Рисунок 9. Результат сборки

В результате разобраны основные понятия при работе со сборкой Webpack. Так же приведёт пример работы css файла через js сборку. Теперь через одну точку входа мы можем подключать неограниченное количество различных файлов и контролировать связь между ними. Данную работу можно использовать как основу для дальнейшего изучения инструмента.

Библиографический список

1. Филонов Д.Р., Тупикин В.И. Чат-бот для Telegram для помощи абитуриентам //Заметки по информатике и математике Сборник научных статей. Ярославль, 2017. С. 152-156.
2. Штыков А.А. Современные тенденции и стандарты в области организации труда front-end разработчика // Проблемы и перспективы студенческий науки. 2017. № 2 (2). С. 46-47.
3. Перфильев Н.В., Макаров Д.А. Оптимизация сборки frontend-приложений // Наука и образование: актуальные исследования и разработки Материалы II Всероссийской научно-практической конференции. Ответственный редактор Н.С. Кузнецова. 2019. С. 44-46.