УДК 004

ИСПОЛЬЗОВАНИЕ WEBPACK ДЛЯ СБОРКИ ПРОЕКТА

Попков И.В. [1], Курзаева Л.В. [2]

[1] *ФГБОУ ВО «Магнитогорский государственный технический университет имени Г.И. Носова»Магнитогорск, Россия email: iliailiaq2@mail.ru.*

[2]*ФГБОУ ВО «Магнитогорский государственный технический университет имени Г.И. Носова» Магнитогорск, Россия email: lkurzaeva@mail.ru.*

В настоящее время набирают популярность приложения, у которых клиентская часть является полнофункциональной. То есть, если раньше клиентская часть приложения отвечала только за вывод информации, который генерируется и отдается сервером, то теперь клиент способен сам правильном образом генерировать полученную информацию. Такую возможность дает javascript и его фреймворки и библиотеки. Но возникает проблема, так как теперь код, генерирующий страницу передается на клиент, а зачастую этого кода много, то скорость загрузки самой страницы и ее работа существенно замедляется. Решить данную проблему помогают сборщики проектов, которые способны минифицировать и сжать js-код, css и другие файлы для более быстрой загрузки. В этой статье подробно рассмотрен сборщик проектов webpack. Рассмотрена логика работы данного сборщика, его особенности. Описаны основные составляющие данного сборщика, их основное предназначение и способы задания в конфигурационном файле сборщика. Описаны алгоритмы создания конфигурационного файла, описаны способы подключения различных плагинов, которые позволяют проекту для действий и пользовательских преобразований. Также, в статье рассмотрены способы работы с уже собранными пакетами программы, или иначе бандлами, их подключения на страницу. Описана специальная среда, которая необходима webpack для работы.

Ключевые слова: webpack, сборщик проектов, минификация, полнофункциональный клиент.

USE WEBPACK FOR ASSEMBLY OF THE PROJECT

Popkov I.V. [1], Kurzaeva L.V. [2]

[1] *FGBOU VO "Magnitogorsk State Technical University named after G.I. Nosov », Magnitogorsk, Russia email: iliailiaq2@mail.ru.*

[2] *FGBOU VO "Magnitogorsk State Technical University named after G.I. Nosov », Magnitogorsk, Russia email: lkurzaeva@mail.ru.*

Currently, applications are gaining popularity, in which the client part is fully functional. That is, if before the client part of the application is responsible only for the output of information that is generated and given by the server, now the client can monitor the correct information. This possibility is provided by javascript and its frameworks and libraries. But there is a problem, because now the code that generates the page is sent to the client, and often this code is a lot, the download speed of the page and its work is reduced. To solve this problem help project builders who can minify and compress js-code, css and other files for faster loading. This article details the webpack project collector. The logic of the work of this collector, its features is considered. The main components of this collector are described, their main purpose and methods of setting in the collector configuration file. Describes the algorithms for creating a configuration file, describing how to connect various plug-ins that allow the project for actions and user-defined transformations. Also, the article discusses ways to work with the already assembled software packages, or else the bundles, their connections to the page. A special environment is described, which is necessary for the webpack to work

Key words: webpack, project collector, minification, full-featured client..

В последние годы, в индустрии web-разработки набирают популярность Single Page Application(SPA), приложений, которые размещены на одной странице, которое для работы подгружает необходимый код вместе с загрузкой страницы. Также пользуются спросом веб-приложения, в которых клиент является полнофункциональном, что дает возможность снизить нагрузку на сервер. Это положительно сказывается на скорости работы приложения, особенно если уникальных посетителей больше десяти тысяч в месяц. Логика на стороне создается при помощи javascript и его фреймворков. Сервер просто отдает js-код клиенту, но возникает проблема, связанная с долгой загрузкой и обработкой этого кода. В качестве решения данно проблемы используются сборщики проектов, которые минимизируют код, удаляют комментарии в нем, могут загружать этот код не целиком, а только при необходимости. В данной статье будет рассмотрен сборщик проектов webpack, который позволяет решить большую часть проблем.

Ниже описаны основные проблемы связанные с разработкой сложного приложения:

1) Трудно управлять кодовой базой;
2) Сложность управления порядком загрузки большого количеств файлов или модулей;
3) Трудность с управлением структуры приложения;
4) Управление зависимостями в различных js-файлов.
5) Большое количество файлов в различных расширениях, которые необходимо компилировать и динамически подключать к странице.

На рисунке 1 представлена логика работы webpack.

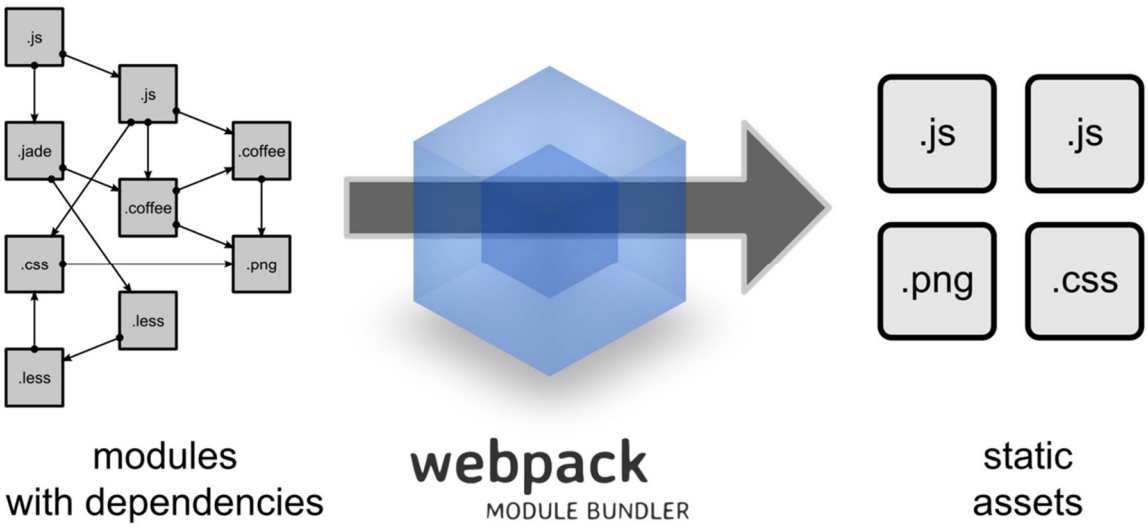


Рисунок 1 – Логика работы webpack

Основная идея, взять все исходные файлы, css, js, sass,less, png и т.д. и преобразовать их в пакеты, называемые как бандлы. После чего эти бандлы подключаются на страницу приложения и приложение работает. Данные бандлы, зачастую, минифицированы, то есть сжаты по размеру, за счет удаления комментариев в коде и минификации, что приводит к ускорению загрузки этих файлов. Создание бандлов осуществляется при помощи построение графов зависимости.

Каждый раз, когда один файл зависит от другого, то есть в один файл импортируется другой файл, webpack рассматривает это как зависимость. Это позволяет webpack принимать файлы не относящиеся к javascript-коду, например, картинки, шрифты.

Когда webpack обрабатывает приложение, оно начинается с списка модулей, определенных в командной строке или в файле конфигурации. Начиная с этих точек входа, webpack рекурсивно строит график зависимостей, который включает в себя каждый модуль, который требуется вашему приложению, затем упаковывает все эти модули в небольшое количество пакетов - часто, только один - для загрузки браузером.

Основной понятия webpack состоит из четырех элементов, они представлены ниже в таблице 1. Эти понятия являются основой вебпака, зная которые можно его настроить. [1]

Таблица 1 – Основной элементы концепта webpack

Название элемента	Его функции
Entry, точка входа.	Точка входа показывает webpack, какой модуль он должен использовать для построения граф зависимостей приложения, указывается при помощи пути к файлу, запускающего ваше приложение. Можно передавать несколько точек входа при помощи массива, это необходимо для создания chunk
Output, вывод	Вывод необходим для сообщения вебпаку куда необходимо разместить готовую сборку проекта. Указывается путь до папки и название файла.
Loaders, лоадеры	Так сложилось, что webpack может обрабатывать только javascript-файлы, а различные css, sass, png и т.д. он не способен обработать. Именно для решения этйо проблемы были придуманы лоадеры, которые преобразовывают файлы в модули, которые можно добавить в граф зависимостей. Лоадеры по сути, просто указывают вебпаку, каким лоадером обработать незнакомый вебпаку файл.[4]
Plugins, плагины	В то время как лоадеры выполняют только преобразования, основанные на типе файла, плагины чаще используются для выполнения действий и пользовательских преобразований на этапе компиляции или частей вашей сборки. У вебпака очень мощная и гибко настраиваемая система плагинов. Множество плагинов доступно вместе со стандартной сборкой вебпака. Для использования плагина его необходимо сначала импортировать, и затем добавить его в массив плагинов.

Для наглядности приведен пример готовой настройки вебпака на рисунке 2.

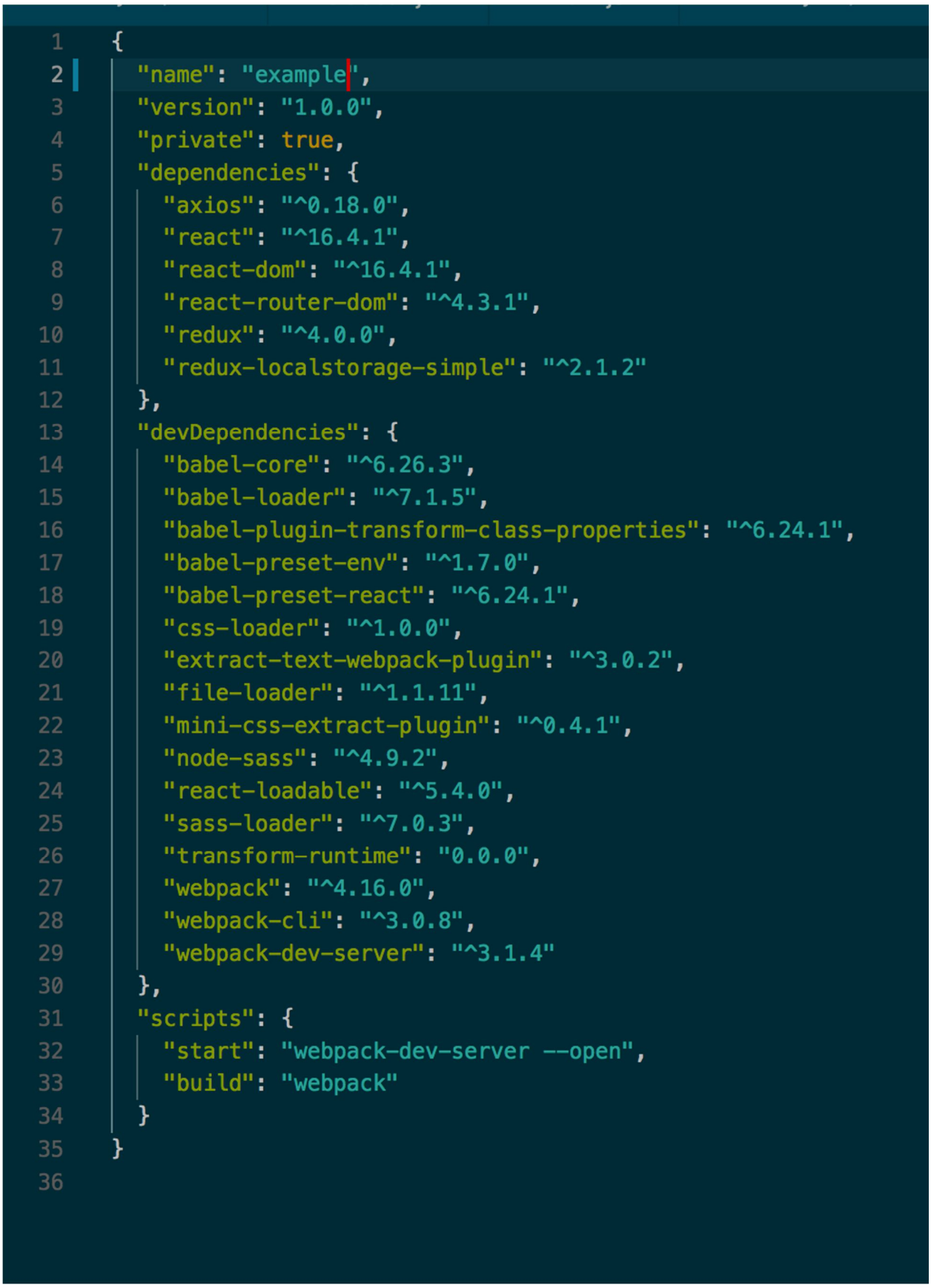
```
const HtmlWebpackPlugin = require('html-webpack-plugin'); //installed
const webpack = require('webpack'); //to access built-in plugins
const path = require('path');

const config = {
  entry: './path/to/my/entry/file.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'my-first-webpack.bundle.js'
  },
  module: {
    rules: [
      { test: /\.txt$/, use: 'raw-loader' }
    ]
  },
  plugins: [
    new webpack.optimize.UglifyJsPlugin(),
    new HtmlWebpackPlugin({template: './src/index.html'})
  ]
};

module.exports = config;
```

Рисунок 2 – Настройка вебпака

Помимо создания конфигурационного файла для стартовой работы webpack необходимо для него развернуть специальное окружение. Окружение создается при помощи Node.js, установка вебпака, пакетов, плагинов, лоадеров происходит при помощи утилиты Node.js.[5] На рисунке 2, при подключении лоадера для обработки файлов в формате txt используется raw-loader, который необходимо поставить заранее, все установленные пакеты и плагины для разработки хранятся в специальном конфигурационном файле package.json. Пример файла приведен на рисунке 3. Этот файл хранит в себе настройки приложения, devDependencies это плагины которые необходимы для разработки продукта, dependencies настройки проекта для использования его пользователями, scripts хранит в себе команды для webpack.[2]



```
{
  "name": "example",
  "version": "1.0.0",
  "private": true,
  "dependencies": {
    "axios": "^0.18.0",
    "react": "^16.4.1",
    "react-dom": "^16.4.1",
    "react-router-dom": "^4.3.1",
    "redux": "^4.0.0",
    "redux-localstorage-simple": "^2.1.2"
  },
  "devDependencies": {
    "babel-core": "^6.26.3",
    "babel-loader": "^7.1.5",
    "babel-plugin-transform-class-properties": "^6.24.1",
    "babel-preset-env": "^1.7.0",
    "babel-preset-react": "^6.24.1",
    "css-loader": "^1.0.0",
    "extract-text-webpack-plugin": "^3.0.2",
    "file-loader": "^1.1.11",
    "mini-css-extract-plugin": "^0.4.1",
    "node-sass": "^4.9.2",
    "react-loadable": "^5.4.0",
    "sass-loader": "^7.0.3",
    "transform-runtime": "0.0.0",
    "webpack": "^4.16.0",
    "webpack-cli": "^3.0.8",
    "webpack-dev-server": "^3.1.4"
  },
  "scripts": {
    "start": "webpack-dev-server --open",
    "build": "webpack"
  }
}
```

Рисунок 3 – Кофигурационный файл package.json

Стоить отметить команду build в scripts с рисунка 3. Эта команда начинает сборку всего проекта, исходя из конфигурации самого webpack и

настроек в package.json. После того, как проект будет собран, в директории, которая указана в output с рисунка 1, появятся все бандлы, то есть собранные js-файлы, css, картинки и другие виды файлов. После чего их можно спокойно подключать на страницу приложения. [3]

В заключении стоит отметить, что сборка проекта при помощи webpack достаточно сложна при первом знакомстве. Но разобравшись с основами можно собирать готовые проекты очень быстро. Благодаря тому, что webpack обладает возможностью собирать проект из разных точек входа, это дает возможность создавать модульность приложения, положительно влияет на скорость приложения, и позволяет разбивать приложение на отдельные куски, которые можно продавать отдельно. Webpack обладает удобной и простой расширяемостью, то есть можно легко менять сборку проекта, добавив и включив плагин в конфигурацию webpack. Именно эти причины делают webpack таким популярным.

Список литературы

1. Официальная документация по webpack [Электронный ресурс] – Режим доступа: https://webpack.js.org/
2. Пособие по webpack [Электронный ресурс] – Режим доступа: https://habr.com/post/309306/
3. Open Source Reference Library [Электронный ресурс] / Apple компания-разработчик мобильной ОС. Режим доступа http://opensource.apple.com/
4. Scruminc [Электронный ресурс] – Режим доступа: https://www.scruminc.com.
5. Модульный frontend при помощи Webpack – Режим доступа: https://learn.javascript.ru/screencast/webpack