**Webpack**

# Установка

**npm i -g webpack**

# Конфигурация

## Простейшая

**webpack.config.js**

**'use strict'**;  
  
module.**exports** = {  
 **entry**: **'./home'**,  
 **output**: {  
 **filename**: **'build.js'** }  
};

## Автоматическая пересборка при изменении файлов

**'use strict'**;  
  
module.**exports** = {  
 **entry**: **'./home'**,  
 **output**: {  
 **filename**: **'build.js'**,  
 **library**: **'home'** },  
  
 **watch**: **true**,  
  
 watchOptions: { *// Опция для ускорения пересборки* **aggregateTimeout**: 100 *// default: 300* }  
};

## Настройка поиска модулей и лоадеров

**resolve**: {  
 **modulesDirectories**: [**'node\_modules'**], *// Где искать модули* **extensions**: [**''**, **'.js'**] *// С какими расширениями*},  
  
**resolveLoader**: {  
 **modulesDirectories**: [**'node\_modules'**], *// Где искать лоадеры* **moduleTemplates**: [**'\*-loader'**, **'\*'**], *// Шаблон поиска* **extensions**: [**''**, **'.js'**] *// С какими расширениями*}

# Сборка

**webpack**

Конфигурация берется из файла **“webpack.config.js”**

## Автоматическая пересборка

**webpack --watch**

## Информация о модулях при сборке

**webpack --display-modules** – показывает в какие файлы пошли модули при сборке

**webpack --display-modules -v** – более подробная информация о модулях при сборке

## Информация на сайте

**webpack --json --profile >stats.json** – получение файла со статистикой

<http://webpack.github.io/analyse/> - сайт для анализа файла

Информация со временем сборки и причинами включения модулей

**webpack --profile --display-modules --display-reasons**

## Сборка нескольких модулей в один файл

**entry**: {  
 **home**: **'./home'**,  
 **about**: **'./about'**,  
 **common**: [**'./welcome'**, **'./common'**] *// Экспортируется только последний*}

## Мультикомпиляция

Используется когда нужно сразу сделать несколько сборок, например с разными параметрами переменной окружения или языка.

***module***.exports = [{*/\* Конфигурация 1 \*/*}, {*/\* Конфигурация 2 \*/*}];

## Ускорение сборки

**webpack.config.js**

**module**: {  
 **loaders**: [{  
 **test**: /\.js$/,  
 **exclude**: /(node\_modules|bower\_components)/, *// Отключение преобразования бабалем зависимостей*

**include**: \_\_dirname + **'/frontend'**, *// Включение только той директории, которую нужно обрабатывать (можно массив указывать)*

**loader**: **'babel'**,  
 **query**: {  
 **presets**: [**'es2015'**],  
 }  
 }],  
  
 **noParse**: /angular\\angular.js/ *// Отключение парсинга больших библиотек, состоящих из одного файла*}

# Статический сервер

## Установка

**npm i -g node-static**

## Запуск в текущей директории

**static &**

# Обращение к модулями с других скриптов

## home.js

**'use strict'**;  
  
**let** welcome = *require*(**'./welcome'**);  
  
welcome(**'home'**);  
  
exports.welcome = welcome;

## webpack.config.js

**'use strict'**;  
  
module.**exports** = {  
 **entry**: **'./home'**,  
 **output**: {  
 **filename**: **'build.js'**,  
 **library**: **'home'** }  
};

## home.html

…

<**script src="build.js"**></**script**>  
</**head**>  
<**body**>  
  
<**script**>home.welcome(**'something?'**)</**script**>  
  
</**body**>  
</**html**>

# Отладка при разработке

## devtool ([link](http://webpack.github.io/docs/configuration.html#devtool))

**eval** – самый быстрый способ построения source-map, потому что его как такового не существует. Используется только для разработки

**cheap-inline-module-source-map** – тоже подходит для разработки

**cheap-source-map** – подходит для production

**source-map** – самый медленный способ построения source-map, используется на production

# Плагины

## Определение переменных

### webpack.config.js

**'use strict'**;  
  
**const** NODE\_ENV = process.**env**.**NODE\_ENV** || **'development'**;  
**const** *webpack* = *require*(**'webpack'**);  
  
 **watch**: NODE\_ENV == **'development'**,  
  
 **devtool**: NODE\_ENV == **'development'** ? **'cheap-inline-module-source-map'** : **null**,  
  
 **plugins**: [  
 **new** *webpack*.DefinePlugin({ **NODE\_ENV**: **JSON**.stringify(NODE\_ENV) })  
 ]  
  
};

### welcome.js

**'use strict'**;  
  
module.exports = **function**(message) {  
  
 **if** (NODE\_ENV == **'development'**) {  
 ***console***.log(message);  
 }  
  
 alert(process.**env**.USER);  
  
 alert(**`Welcome** ${message}**`**);  
};

## Вынос общего функционала

**webpack.config.js**

Если плагин не настраивать, то он вынесет код, который используется во всех точках входа.

**plugins**: [  
 **new** *webpack*.**optimize**.CommonsChunkPlugin({  
 **name**: **'common'**,  
 *//minChunks: 2 // Мин. кол-во модулей, необходимое для выноса общего функционала* **chunks**: [**'about'**, **'home'**] *// Выносит общий функционал для описанных модулей. Для других файлов, плагин можно использовать еще раз* })  
],

## Замена контекста (чтобы убрать лишние модули)

**plugins**: [  
 **new** *webpack*.ContextReplacementPlugin(/node\_modules\\moment\\locale/, /ru|en-gb/)  
],

## Игнорирование модулей при сборке

**plugins**: [  
 **new** *webpack*.IgnorePlugin(/\.\/locale/)  
],

## Удаление старых файлов при сборке

**const** *rimraf* = *require*(**'rimraf'**);

…

{ *// Кастомный плагин, удаляет файлы от сборки, перед новой сборкой* apply: (compiler) => {  
 *rimraf*.sync(compiler.**options**.**output**.**path**);  
 }  
},

# Минификация ([ссылка](https://www.youtube.com/watch?v=ZLrzAhhbt6s&index=9&list=PLDyvV36pndZHfBThhg4Z0822EEG9VGenn))

## webpack.config.js

**if** (NODE\_ENV == **'production'**) {  
 module.exports.**plugins**.push(  
 **new** *webpack*.**optimize**.UglifyJsPlugin({  
 **compress**: {  
 **warnings**: **false**,  
 **drop\_console**: **true**,  
 **unsafe**: **true** }  
 })  
 )  
}

# Динамическая загрузка модулей

## 2 способа

**'use strict'**;  
  
**document**.getElementById(**'loginButton'**).onClick = **function**() {  
  
 *// Способ 1 (require.ensure)  
 require*.ensure([**'./login'**], **function**() {  
 **let** login = *require*(**'./login'**);  
 login();  
 });  
  
 *// Способ 2 (AMD)  
 require*([**'./login'**], **function**(login) {  
 login();  
 });  
   
};

## Упрощенный способ

*require*.ensure([], **function**() {  
 **let** login = *require*(**'./login'**);  
 login();  
});

## Объединение фрагментов

3-й параметр функции **require.ensure**, указывает в какой файл вставить модуль.

**'use strict'**;  
  
**document**.getElementById(**'loginButton'**).**onclick** = **function**() {  
 *require*.ensure([], **function**() {  
 **let** login = *require*(**'./login'**);  
 login();  
 }, **'auth'**);  
};  
  
**document**.getElementById(**'logoutButton'**).**onclick** = **function**() {  
 *require*.ensure([], **function**() {  
 **let** logout = *require*(**'./logout'**);  
 logout();  
 }, **'auth'**);  
};

## Загрузка модуля по названию передаваемому в адресе

Модули хранятся в папке **routes**.

**'use strict'**;  
  
**let** moduleName = **location**.**pathname**.slice(1);***console***.log(**'moduleName:'**, moduleName);  
  
**let** route = *require*(**'./routes/'** + moduleName);  
route();

### Продвинутые require - require.context

**'use strict'**;  
  
**let *moduleName*** = **location**.**pathname**.slice(1); *// /home****console***.log(**'moduleName:'**, ***moduleName***);  
  
**let *context*** = *require*.**context**(**'./routes'**, **false**, /\.js$/);  
  
***context***.keys().forEach(**function**(path) {  
 **let** module = ***context***(path);  
 module();  
});  
  
**try** {  
 **let *route*** = ***context***(**'./'** + ***moduleName***);  
 ***route***();  
} **catch**(e) {  
 alert(e);  
}

# Внешние библиотеки

## Экспортирование для использования в модулях

**webpack.config.js**

**externals**: {  
 **lodash**: **'\_'**}

## Экспортирование модулей в глобальные переменные

**webpack.config.js**

**plugins**: [  
 **new** *webpack*.ProvidePlugin({  
 **map**: **'lodash/map'**,  
 **\_**: **'lodash'** })  
]

## Упрощение доступа к скриптам

**webpack.config.json**

**resolve**: {  
 **root**: \_\_dirname + **'/vendor'**, *// Задает директорию для поиска скриптов помимо node\_modules* **alias**: {  
 **old**: **'old/dist/old'** *// Задает алиас "old" вместо "old/dist/old" для использования модуля* }  
}

## Импортирование переменных в скрипты

Используется модуль "**imports-loader**".

**let** old = *require*(**'imports?workSettings=>{delay:500}!old'**);

## Экспортирование старого скрипта в модуль

Используется модули "**imports-loader**" и "**exports-loader**".

**let** old = *require*(**'imports?workSettings=>{delay:500}!exports?Work!old'**);

## Экспортирование модуля в глобальную переменную (у меня не сработало)

Используется модуль “**expose**”.

module: {  
 **loaders**: [{  
 **test**: /old.js$/,  
 **loader**: **"expose?Work!imports?workSettings=>{delay:500}!exports?Work"** }],  
},

## script-loader

Запуск старого скрипта через **eval**.

module: {  
 **loaders**: [{  
 **test**: /old.js$/,  
 **loader**: **'script'** }],  
},

# Loaders

## Babel

**module**: {  
 **loaders**: [{  
 **test**: /\.js$/,  
 **include**: path.join(\_\_dirname, **'/frontend'**),  
 **loader**: **'babel?presets[]=es2015'**,  
 }],  
}

## CSS и file loaders

**module**: {  
 **loaders**: [{  
 **test**: /\.css$/, *// loader для стилей* **loader**: **'style-loader!css-loader!postcss-loader'** }, {  
 **test**: /\.(png|jpg|svg|ttf|eot|woff|woff2)$/, *// loader для файлов* **loader**: **file?name=[path][name].[ext'** }],  
},  
**postcss**: [ autoprefixer({ **browsers**: [**'last 2 versions'**] }) ]

## URL loader

**module**: {  
 **loaders**: {  
 **test**: /\.(png|jpg|svg|ttf|eot|woff|woff2)$/, *// loader для файлов, при длине файла < 4096 байт, используется data url* **loader**: **'url?name=[path][name].[ext]&limit=4096'** }],  
}

## Jade loader ([ссылка](https://www.youtube.com/watch?v=6WxoxUuSVA4&list=PLDyvV36pndZHfBThhg4Z0822EEG9VGenn&index=32))

**module**: {  
 **loaders**: [{  
 **test**: /\.jade$/,  
 **loader**: **'jade'** }],  
}

## Stylus loader

Используются библиотеки: **stylus** и **stylus-loader**.

Опция **resolve url** правильно устанавливает url у файлов, которые импортируются с других файлов стилей.

Опция **linenos** добавляет в виде комментариев номера строк и пути к файлам, откуда взяты стили.

**module**: {  
 **loaders**: {  
 **test**: /\.styl$/,  
 **loader**: **'style-loader!css-loader!postcss-loader!stylus?resolve url'** }],  
}

## LESS loader

Опция **resolve url** подключена по умолчанию.

**module**: {  
 **loaders**: {  
 **test**: /\.styl$/,  
 **loader**: **'style-loader!css-loader!postcss-loader!less'** }],  
}

## Сохранение стилей в файл

### Вынос только статически подключаемых стилей

Используется модуль: **extract-text-webpack-plugin**.

**let** *ExtractTextPlugin* = *require*(**'extract-text-webpack-plugin'**);

**…**

**module**: {  
 **loaders**: [{  
 **test**: /\.styl$/,  
 **loader**: *ExtractTextPlugin*.extract(**'style'**, **'css!stylus?resolve url&linenos'**) *// 1-й арг-т - это лоадер если стили остаются внутри JS (например если какой-то скрипт подкл-ся динамически)*

}, {  
 **test**: /\.(png|jpg|svg|ttf|eot|woff|woff2)$/,  
 **loader**: **'file?name=[path][name].[ext]'** }],  
},  
  
**plugins**: [  
 **new** *ExtractTextPlugin*(**'styles.css'**)  
]

### Вынос всех стилей

**let** *ExtractTextPlugin* = *require*(**'extract-text-webpack-plugin'**);

**…**

**module**: {  
 **loaders**: [{{  
 **test**: /\.styl$/,  
 **loader**: *ExtractTextPlugin*.extract(**'css!stylus?resolve url&linenos'**)  
 }, {  
 **test**: /\.(png|jpg|svg|ttf|eot|woff|woff2)$/,  
 **loader**: **'file?name=[path][name].[ext]'** }],  
},  
  
**plugins**: [  
 **new** *ExtractTextPlugin*(**'styles.css'**, { **allChunks**: **true** }) *// Вытаскивает стили в файл со всех скриптов, включая динамически подргужаемые*]

## Файлы стилей для каждой точки входа

Вместо имени файла указывается **[name]**.

***module***.exports = {  
 **context**: path.join(\_\_dirname, **'/frontend'**),  
 **entry**: {  
 **main**: **'./main'**,  
 **hello**: **'./hello'** },  
   
 **module**: {  
 **loaders**: [{  
 **test**: /\.styl$/,  
 **loader**: *ExtractTextPlugin*.extract(**'css!stylus?resolve url&linenos'**)  
 }, {  
 **test**: /\.(png|jpg|svg|ttf|eot|woff|woff2)$/,  
 **loader**: **'file?name=[path][name].[ext]'** }],  
 },  
  
 **plugins**: [  
 **new** *ExtractTextPlugin*(**'****[name].css'**, { **allChunks**: **true** }) *// Вытаскивает стили в файл со всех скриптов, включая динамически подргужаемые* ]  
};

# Кэширование

## Настройки для кэширования

**[hash]** – хэш всей компиляции (внутри webpack, у плагинов свои параметры)

**[chunkhash]** – хэш файла (внутри webpack, у плагинов свои параметры)

**output**: {  
 **path**: path.join(\_\_dirname, **'/public/assets'**),  
 **publicPath**: **'/assets/'**,  
 **filename**: **'[name].[chunkhash]js'**,  
 **chunkFilename**: **'[id].js'**,  
 **library**: **'[name]'**},

**new** *ExtractTextPlugin*(**'[name].[contenthash].css'**, { **allChunks**: **true** })

## Генерация assets.json

Файл из которого бэкэнд может взять имя файла с хэшэм и сгенерировать html страницу.

**plugins**: [  
 **new** *AssetsPlugin*({  
 **filename**: **'assets.json'**,  
 **path**: \_\_dirname + **'/public/assets'** })  
]

## Генерация html-файла

**new** *HtmlWebpackPlugin*({  
 **filename**: **'./about.html'**, *// Выходной файл* **chunks**: [**'common'**, **'about'**] *// Точки входа*})

## Включение хэша только в production

**function** *addHash*(template, hash) {  
 **return** NODE\_ENV == **'production'** ? template.**replace**(/\.[^.]+$/, **`.[**${hash}**]$&`**) : template;  
}

…

**filename**: *addHash*(**'[name].js'**, **'chunkhash'**),  
**chunkFilename**: *addHash*(**'[id].js'**, **'chunkhash'**),

## Использование хэша при обращении к файлу в development

**function** *addHash*(template, hash) {  
 **return** NODE\_ENV == **'production'** ? template.**replace**(/\.[^.]+$/, **`.[**${hash}**]$&`**) : **`**${template}**?hash=[**${hash}**]`**;  
}

При генерации html получится:

<**link href="/assets/home.css?hash=c7246ad90855cfb724ade8e0d6c73f19" rel="stylesheet"**></**head**>

# Live Reload

## Webpack Dev Server

### Установка

npm i -g webpack-dev-server

### Простейшая конфигурация

**'use strict'**;  
  
**const** path = *require*(**'path'**);  
  
module.exports = {

…

**devServer**: {  
 **host**: **'localhost'**, *// default* **port**: 8080, *// default* **contentBase**: path.join(\_\_dirname, **'backend'**) *// Директория откуда брать контент* }  
};

### Конфигурация с proxy-сервером

**devServer**: {  
 **host**: **'localhost'**, *// default* **port**: 8080, *// default* **proxy**: [{  
 **path**: /.\*/,  
 **target**: **'http://localhost:3000'** }]  
}

## Webpack Dev Middleware ([link](https://youtu.be/1BnRKzlwa7M?list=PLDyvV36pndZHfBThhg4Z0822EEG9VGenn&t=596))

<https://github.com/webpack/webpack-dev-middleware>

Модуль, который встраивается в приложение написанное на Express (NodeJS).

## Live Reload

### Конфигурация

У файлов обязательно должен быть **Hash**, иначе браузер не узнает об изменении файлов и не обновит их.

***module***.exports = {  
 **context**: path.join(\_\_dirname, **'frontend'**),  
 **entry**: {  
 **main**: [**'webpack-dev-server/client'**, **'webpack/hot/dev-server'**, **'./main'**]  
 },  
 **output**: {  
 **path**: path.join(\_\_dirname, **'public'**),  
 **publicPath**: **'/'**,  
 **filename**: **'[name].js'**,  
 **library**: **'[name]'** },  
  
 **module**: {  
 **loaders**: {  
 **test**: /\.(png|jpg|svg|ttf|eot|woff|woff2)$/,  
 **loader**: **'file?name=[path][name].[ext]'** *// Обновление благодаря hash* }],  
 },  
  
 **plugins**: [  
 **new** webpack.HotModuleReplacementPlugin()  
 ],  
  
 **devServer**: {  
 **contentBase**: path.join(\_\_dirname, **'backend'**),  
 **hot**: **true** }  
};

### Параметры командной строки

**--inline** - добавит точку входа **'webpack-dev-server/client'**

**--hot** – добавит точку входа **'webpack/hot/dev-server'** и плагин **webpack.HotModuleReplacementPlugin**

### Отключение плагина ExtractTextPlugin в режиме разработки

**plugins**: [  
 **new** *ExtractTextPlugin*(**'[name].css'**, { **allChunks**: **true**, **disable**: process.**env**.**NODE\_ENV** == **'development'** })]

## Hot Module Replacement

Модуль, преобразованный таким образом, чтобы обновлять меню при его изменении без перезагрузки страницы.

**frontend/main.js**

**'use strict'**;  
  
**let *Menu***;  
**let *pandaMenu***;  
  
**function** *showMenu*() {  
 ***Menu*** = *require*(**'./menu'**).**default**;  
 ***pandaMenu*** = **new *Menu***({  
 **title**: **'Меню панды'**,  
 **items**: [{  
 **text**: **'Яйца'**,  
 **href**: **'#eggs'** }, {  
 **text**: **'Мясо'**,  
 **href**: **'#meat'** }, {  
 **text**: **'99% еды - бамбук.1'**,  
 **href**: **'#bamboo'** }]  
 });  
 **document**.**body**.appendChild(***pandaMenu***.**elem**);  
}  
*showMenu*();  
  
**if** (***module***.hot) { *// Для того, чтобы в продакшэне код удалился* ***module***.hot.accept(**'./menu'**, () => {  
 **document**.**body**.removeChild(***pandaMenu***.**elem**);  
 *showMenu*();  
 });  
}

## react-hot-loader ([link](https://youtu.be/EQhXtTOxpVk?list=PLDyvV36pndZHfBThhg4Z0822EEG9VGenn&t=910))

<https://github.com/gaearon/react-hot-loader>

Модуль для Reac.js, перезагружает компоненты без перезагрузки приложения.