РАБОТА С WEBPACK ОСНОВЫ УПРАВЛЕНИЯ МОДУЛЯМИ

JavaScript

Модуль 3. Урок 1.

LUXOFT

LXFT LISTED NYSE

МОДУЛИ JAVASCRIPT

Зачем нужны модули?
Когда приложение сложное и кода много – мы пытаемся разбить его на файлы. В каждом файле описываем какую-то часть, а в дальнейшем – собираем эти части воедино.

Что такое модуль?
Модулем считается файл с кодом.

В этом файле ключевым словом export помечаются переменные и функции, которые могут быть использованы снаружи.

Другие модули могут подключать их через вызов import.

ЭКСПОРТ ИЗ МОДУЛЯ

Ключевое слово export можно ставить:

- перед объявлением переменных, функций и классов
- отдельно, при этом в фигурных скобках указывается, что именно экспортируется

// экспорт прямо перед объявлением
export let one = 1;

Можно написать **export** и отдельно от объявления:
let two = 2;
export {two};

Для двух переменных будет так:

export {one, two};

При помощи ключевого слова as можно указать, что переменная one будет доступна снаружи (экспортирована) под именем once, а two – под именем twice:

export {one as once, two as twice};

ЭКСПОРТ ФУНКЦИЙ И КЛАССОВ

Экспорт функций и классов выглядит так же:

```
export class User {
  constructor(name) {
    this.name = name;
  }
};

export function sayHi() {
  alert("Hello!");
};

// отдельно от объявлений было бы так:
// export {User, sayHi}
```

```
// функция без имени − так будет ошибка:
export function() { alert("Error"); };
```

ИМПОРТ

Другие модули могут подключать экспортированные значения при помощи ключевого слова import.

Синтаксис:
import {one, two} from "./nums";

Здесь:

- "./nums" – модуль, как правило это путь к файлу модуля.
- one, two – импортируемые переменные, которые должны быть обозначены в nums словом export.

В результате импорта появятся локальные переменные one, two, которые будут содержать значения соответствующих экспортов.

Импортировать можно и под другим именем, указав его в «**as**»:

```
// импорт one под именем item1, a two – под именем item2
import {one as item1, two as item2} from "./nums";

alert( `${item1} and ${item2}` ); // 1 and 2
```

ИМПОРТ ВСЕХ ЗНАЧЕНИЙ СРАЗУ

Можно импортировать все значения сразу в виде объекта вызовом **import * as obj**, например:

```
import * as numbers from "./nums";

// теперь экспортированные переменные - свойства numbers
alert( `${numbers.one} and ${numbers.two}` ); // 1 and 2
```

МЕНЕДЖЕР ПАКЕТОВ NPM

Node Package Manager (NPM) - менеджер пакетов Node

NPM предоставляет следующие возможности:

- Онлайн-репозитории для пакетов/модулей Node.js
- Утилита командной строки для установки пакетов Node.js packages, менеджмента версий и зависимостей

PACKAGE.JSON

Для управления установленными библиотеками используется package.json.
Начальную версию можно создать с помощью **npm init.**
Далее при установке пакетов командой npm install <package_name>
Библиотека автоматически добавляется в package.son

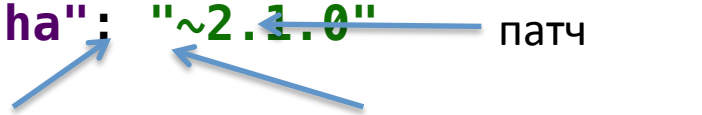
Пример **package.json:**

```
{
  "name": "async-lib",
  "version": "1.1.2",
  "description": ”Async library",
  "main": "index.js",
  "scripts": {
    "test": "mocha test.js"
  },
  "author": "Vladimir Sonkin",
  "license": "ISC",
  “keywords": “async",
```

ВЕРСИОНИРОВАНИЕ В NPM

Пример **package.json (продолжение):**

```
  "dependencies": {
    "bluebird": "^3.5.0"
  },
  "devDependencies": {
    "mocha": "~2.1.0"
  }
}
```



^ означает, что возможно обновление до последней минорной версии
например ^3.5.0 может обновиться до 3.7.1, но не до 4.*

~ означает, что возможно обновление до последнего патча
например, ~2.1.0 может обновиться до 2.1.3, но не до 2.2.*

Мажорная версия – значительные изменения библиотеки, без гарантии обратной совместимости
Минорная версия – незначительные изменения, с гарантией обратной совместимости
Патч – исправление ошибок

КОМАНДЫ NPM

Для примера используется пакет express – но это может быть любая доступная библиотека.

- **npm init** – создать npm в диалоговом режиме
- **npm install** – установить все зависимости из package.json
- **npm install express** – установить express: скачать его в папку node_modules, добавить запись в package.json
- **npm uninstall express** – удалить express
- **npm install express@3.2.1** - установить express определенной версии
- **npm search express** – найти express в репозитории
- **npm update express** – обновить express до последней минорной версии
- **npm install webpack –g** – установить webpack глобально
- **npm adduser** – добавить пользователя npm
- **npm publish** – опубликовать собственную библиотеку

ПАПКИ NPM

Локальная инсталляция: помещает все в папку **./node_modules в корне проекта**

Глобальная инсталляция (с флагом -g): помещает в папку **/usr/local** или в папку, где установлен Node

Установите пакет локально, если вы собираетесь его импортировать.

Установите пакет глобально, если вы собираетесь запускать пакет из командной строки.

Если вам нужно и то, и другое, придется установить дважды (альтернатива – использовать npm link).

PACKAGE-LOCK.JSON

package-lock.json автоматически генерируется для любых операций, где npm изменяет либо дерево node_modules, либо package.json

Этот файл предназначен для передачи в системы контроля версий (git, svn) и выполняет различные задачи:

- Описать единое представление дерева зависимостей
- Предоставьте пользователям возможность «путешествовать по времени» в предыдущие состояния node_modules без необходимости фиксировать сам каталог.
- Для облегчения большей видимости изменений дерева с помощью читаемых различий в управлении версиями.

Таким образом, **не нужно сохранять node_modules в GIT**!

package-lock.json позволяет однозначно восстановить содержимое папки node_modules

ИМПОРТ ИЗ NODE_MODULES И ИЗ ЛОКАЛЬНОГО ФАЙЛА

При импорте возможно 2 варианта синтаксиса:

```
import {one} from "./nums";
или
import {one} from "nums";
```

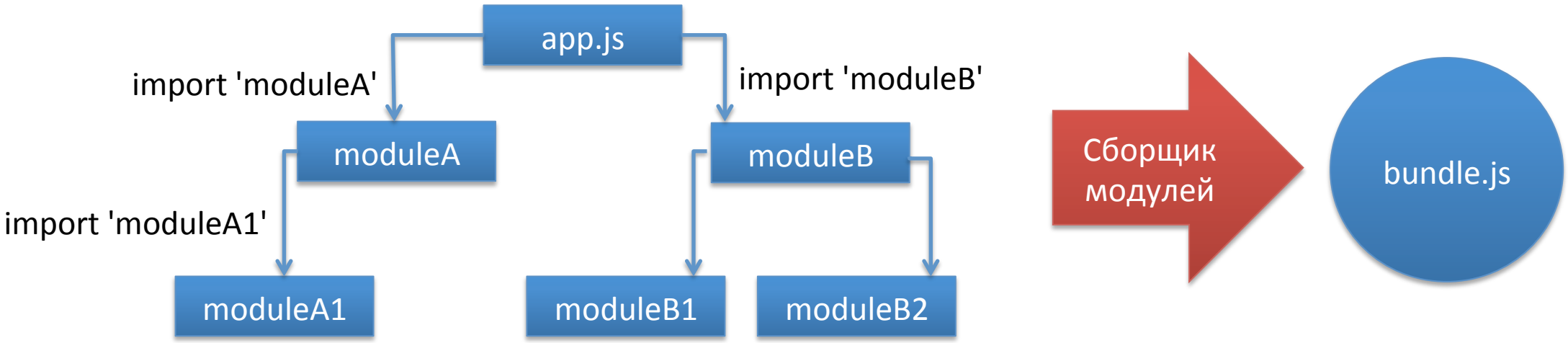
Первый вариант − импорт из локального файла. Здесь будет найден файл относительно текущего файла (в той же папке) и импортирован.

Второй вариант − импорт из папки node_modules. Будет найдена папка nums, в ней − файл index.js и его содержимое будет импортировано.

РАБОТА С МОДУЛЯМИ В БРАУЗЕРЕ

Однако поддержка модулей не реализована в браузерах.
Причина − загрузка модулей по одному приведет к очень долгой загрузке сайта: ведь таких модулей, вместе со всеми требуемыми библиотеками, может быть тысячи!

Что же делать? Можно использовать **сборщик модулей**:

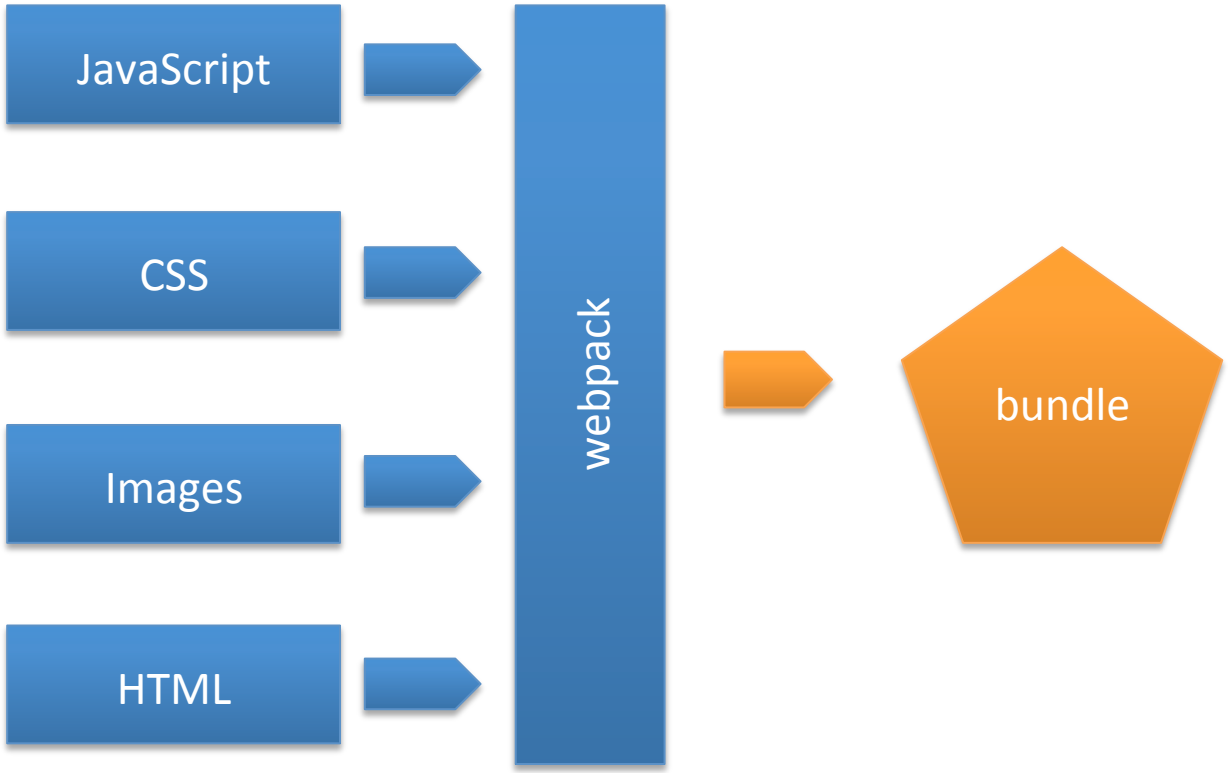


На выходе мы получаем 1 файл, который можно подключить в SPA (одностраничное приложение):
<script src="bundle.js"></script>

При этом он будет включать **все** необходимые **модули** и **все библиотеки**!

ЧТО ТАКОЕ WEBPACK?

Самый популярный сборщик модулей − это Webpack.
Webpack берет модули с зависимостями и генерирует статические ресурсы, которые представляют эти модули.



Когда webpack обрабатывает ваше приложение, он создает граф зависимостей, который отображает каждый модуль, необходимый вашему проекту, и генерирует один или несколько пакетов (bundle).

КАК НАЧАТЬ РАБОТУ С WEBPACK?

Для установки вебпака нужен Node.

Теперь можно установить вебпак глобально:

```
$ npm install -g webpack
```

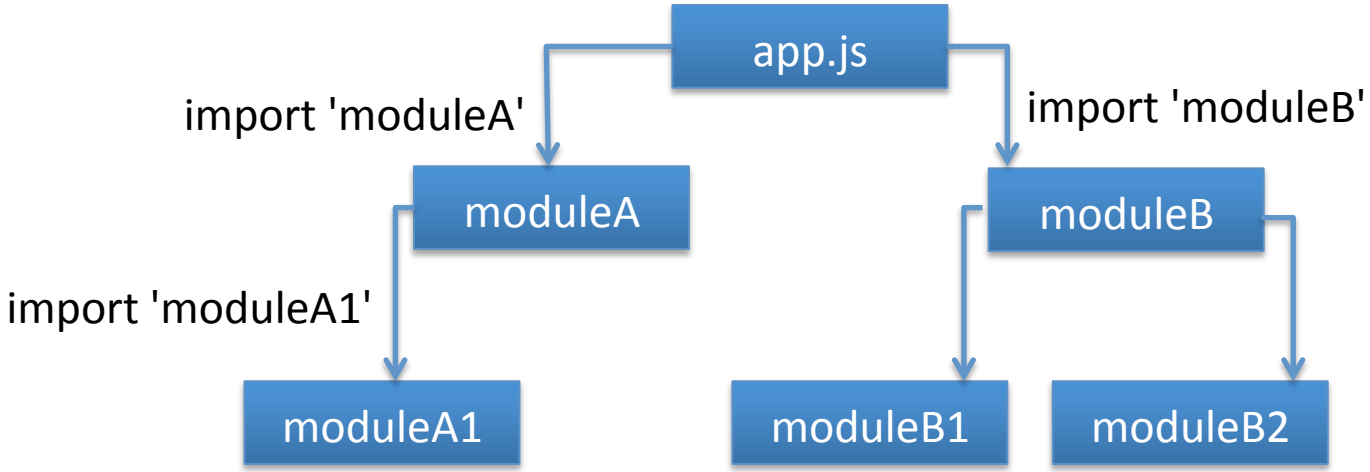
ТОЧКА ВХОДА

Точка входа указывает, какой модуль webpack должен использовать, чтобы начать строить свой внутренний граф зависимостей. Webpack будет определять, какие из других модулей и библиотек зависят от точки входа (прямо и косвенно).

По умолчанию его значение равно ./src/index.js, но вы можете указать другую (или несколько точек входа), настроив свойство записи в конфигурации webpack. Например:

webpack.config.js:

```
module.exports = {
  entry: './app.js'
};
```



ВЫХОДНОЙ ФАЙЛ

Свойство output указывает webpack, где следует испускать пакеты, которые он создает, и имена этих файлов. По умолчанию используется **./dist/main.js** для основного выходного файла и папки **./dist** для любого другого сгенерированного файла.

Вы можете настроить эту часть процесса, указав поле вывода в своей конфигурации:

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

module.exports = {
  entry: './app.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'my-first-webpack.bundle.js'
  }
};
```

ЗАГРУЗЧИКИ

Из коробки webpack понимает только файлы JavaScript и JSON. Загрузчики позволяют webpack обрабатывать файлы других типов и преобразовывать их в допустимые модули, которые могут быть использованы вашим приложением и добавлены в граф зависимостей.

```
module.exports = {
  output: {
    filename: 'my-first-webpack.bundle.js'
  },
  module: {
    rules: [
      { test: /\.txt$/, use: 'raw-loader' }
    ]
  }
};
```

ПРИМЕРЫ ЗАГРУЗЧИКОВ

Вы можете использовать загрузчики, чтобы сообщить webpack, как загрузить файл CSS или преобразовать TypeScript в JavaScript. Для этого вы должны начать с установки необходимых вам загрузчиков:

```
npm install --save-dev css-loader
npm install --save-dev ts-loader
```

Затем сконфигурируйте webpack использовать css-загрузчик для каждого .css-файла и загрузчик ts-loader для всех файлов .ts:

```
module.exports = {
  module: {
    rules: [
      { test: /\.css$/, use: 'css-loader' },
      { test: /\.ts$/, use: 'ts-loader' }
    ]
  }
};
```

КОНФИГУРИРОВАНИЕ ЗАГРУЗЧИКОВ

module.rules позволяет указать несколько загрузчиков в конфигурации вашего веб-пакета. Это краткий способ отображения загрузчиков и помогает поддерживать чистый код. Он также предлагает вам полный обзор каждого соответствующего загрузчика.

Загрузчики выполняются, начиная с последнего. В приведенном ниже примере выполнение начинается с sass-loader, продолжается css-loader и, наконец, заканчивается загрузчиком стилей:

```
module.exports = {
  module: {
    rules: [
      {
        test: /\.css$/,
        use: [
          { loader: 'style-loader' },
          {
            loader: 'css-loader',
            options: {
              modules: true
            }
          },
          { loader: 'sass-loader' }
        ]
      }
    ]
  }
};
```

ВОЗМОЖНОСТИ ЗАГРУЗЧИКОВ

- Загрузчики могут быть выстроены в **цепочки**. Каждый загрузчик в цепочке применяет преобразования к обрабатываемому ресурсу. Цепочка выполняется в **обратном порядке**. Первый загрузчик передает свой результат (ресурс с прикладными преобразованиями) на следующий и т. д. Webpack ожидает, что **в результате** выполнения всей цепочки будет получен **JavaScript**.
- Загрузчики могут быть синхронными или асинхронными.
- Загрузчики работают в **Node.js** и могут делать все, что может Node.
- Загрузчики могут быть настроены с помощью объекта **options**.
- **Плагины** могут предоставить загрузчикам больше возможностей.
- Загрузчики могут создавать **дополнительные файлы**.