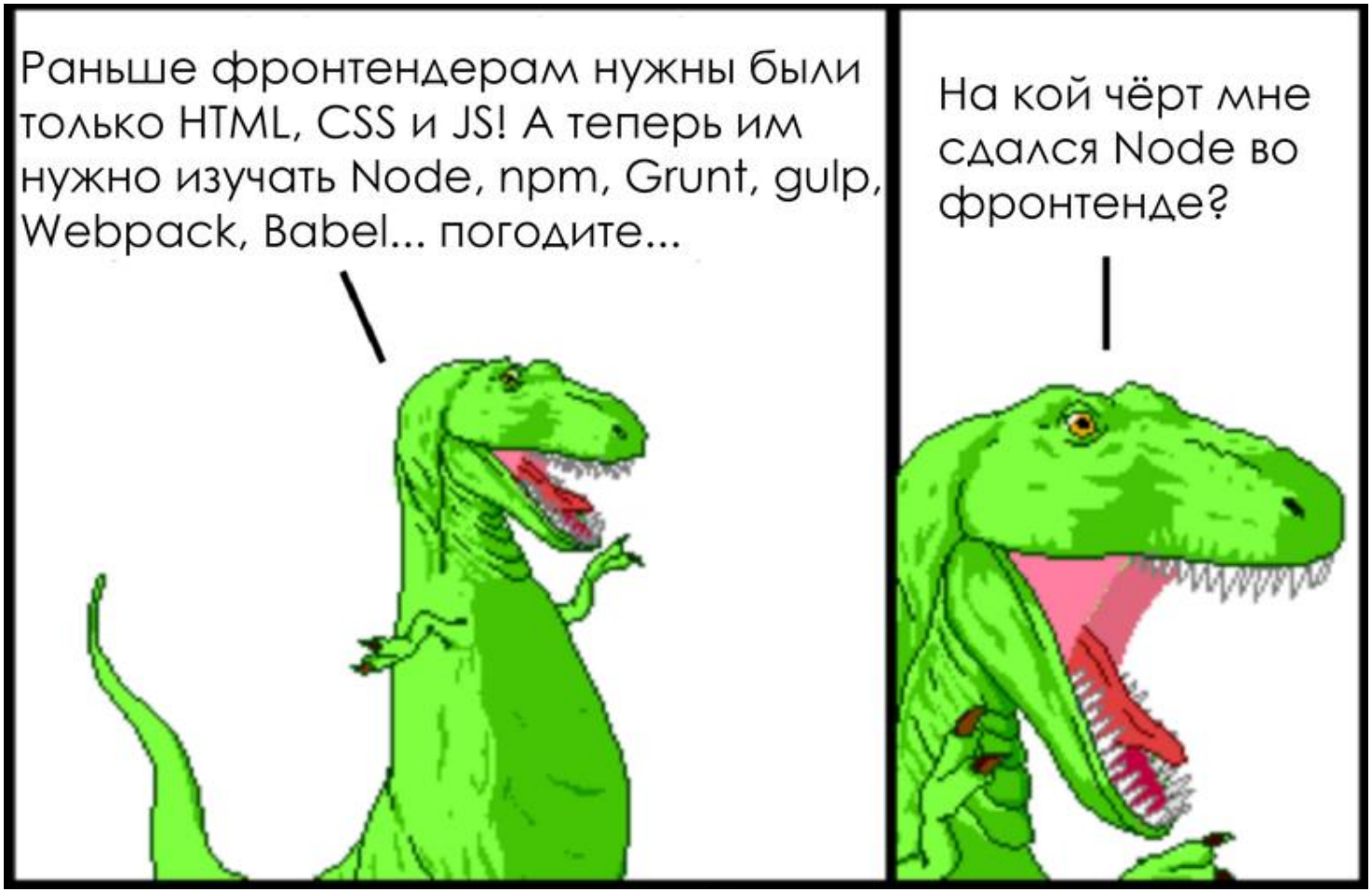
Современный JavaScript

Современный JavaScript

JS



OLD SCHOOL

Old School Approach

JS

- Создать html файл
- Скачать нужный js файл
- Подключить js файл
- Использовать функции из добавленного js файла

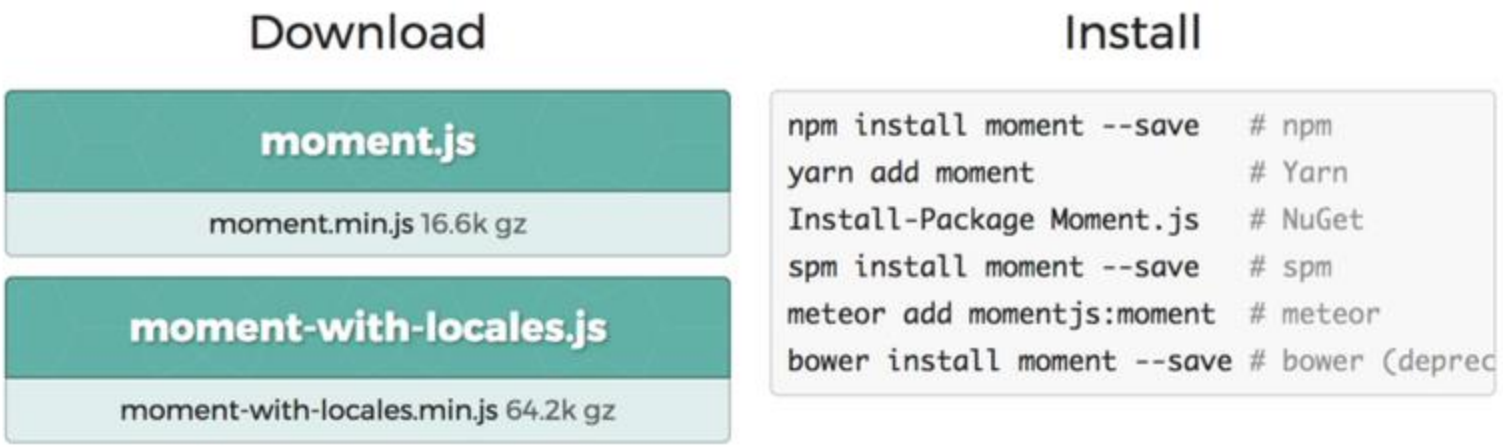
Old School Approach


```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JavaScript Example</title>
  <script src="index.js"></script>
</head>
<body>
  <h1>Hello from HTML!</h1>
</body>
</html>
```

Добавить библиотеку

JS

```
moment().startOf('day').fromNow(); // 20 часов назад
```



Можно добавить moment.js на сайт, скачав файл moment.min.js в директорию и включив его в наш файл index.html.

Old School Approach


```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JavaScript Example</title>
  <script src="moment.min.js"></script>
  <script src="index.js"></script>
</head>
<body>
  <h1>Hello from HTML!</h1>
</body>
</html>
```

npm

USING A JAVASCRIPT PACKAGE MANAGER

NPM Approach

JS

- Создать html файл
- Используя командную строку через npm установить необходимый пакет
  `npm install <package>`
- Подключить js файл
- Использовать функции из добавленного файла

NPM Approach

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JavaScript Example</title>
  <script src="index.js"></script>
</head>
<body>
  <h1>Hello from HTML!</h1>
</body>
</html>
```

NPM Approach


```
$ npm init

package.json
{
   "name": "your-project-name",
   "version": "1.0.0",
   "description": "",
   "main": "index.js",
   "scripts": {
      "test": "echo \"Error: no test specified\" && exit 1"
   },
   "author": "",
   "license": "ISC"
}
```

NPM Approach

```
$ npm install moment --save
```

- Скачивает весь код из пакета moment.js в папку под названием node_modules.
- Автоматически модифицирует файл package.json для отслеживания moment.js в качестве проектной зависимости.

NPM Approach

JS

package.json

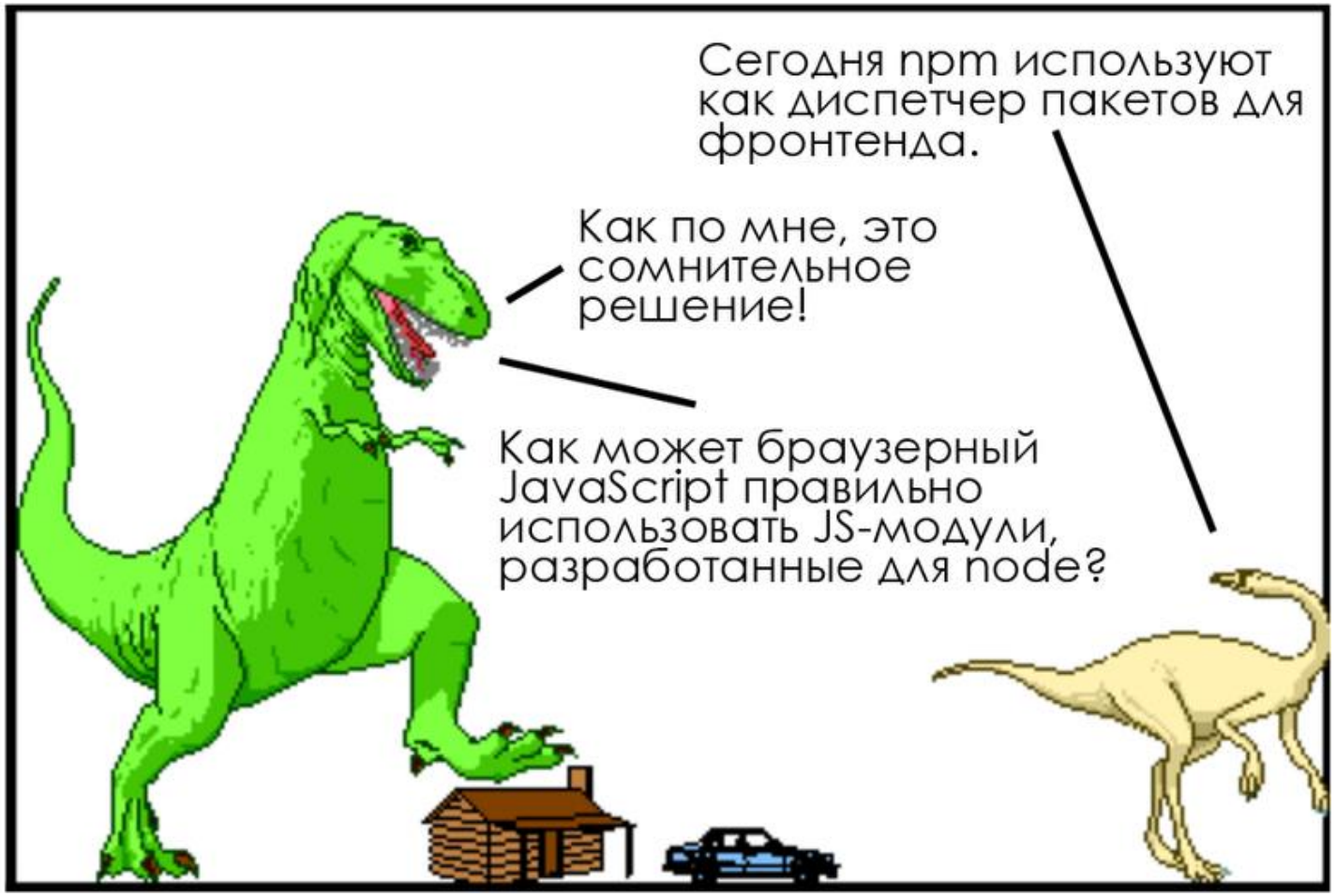
```
{
   "name": "your-project-name",
   "version": "1.0.0",
   "description": "",
   "main": "index.js",
   "scripts": {
      "test": "echo \"Error: no test specified\" && exit 1"
   },
   "author": "",
   "license": "ISC",
   "dependencies": {
      "moment": "^2.19.1"
   }
}
```

Другие разработчики могут автоматически устанавливать нужные пакеты с помощью команды `npm install`

NPM Approach

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JavaScript Example</title>
  <script src="index.js"></script>
  <script
      src="node_modules/moment/min/moment.min.js">
  </script>
</head>
<body>
  <h1>Hello from HTML!</h1>
</body>
</html>
```

NPM Approach



webpack

USING A JAVASCRIPT MODULE BUNDLER

Bundler Approach

JS

- В JS изначально не было импорта, код исполнялся только в браузере
- Для организации JS-кода отдельные файлы загружались с глобально доступными переменными.
- 2009 г. – спецификация CommonJS для экспорта/импорта JS модулей. Используется в node.js
- 2017 г. –Native JS Import

Node.js, пример кода

JS

```
// index.js
var moment = require('moment');
console.log("Hello from JavaScript!");
console.log(moment()
  .startOf('day')
  .fromNow()
);
```

Node.js, пример кода

JS

- Node.js — это серверный язык с доступом к файловой системе.
- Node.js известно расположение npm-модулей.

Вместо

```
require('./node_modules/moment/min/moment.min.js')
```

Можно использовать

```
require('moment')
```

Не работает в браузере! (require не определён)

Bundler Approach

JS

- Bundler - инструмент для сборки модулей в единые пакеты, имеющий доступ к файловой системе
- В JS bundler ищет все **require** и заменяет их на содержимое файлов
- В итоге получается единый JS-файл без выражений **require**

Bundler Approach

JS

- http://browserify.org/ – создан в 2011г., самый популярный бандлером до 2015 г.
- https://webpack.github.io/ - лидер на настоящее время

Webpack use case

JS

- npm install webpack --save-dev

```
{
   "name": "your-project-name",
   "version": "1.0.0",
   "description": "",
   "main": "index.js",
   "scripts": {
      "test": "echo \"Error: no test specified\" && exit 1"
   },
   "author": "",
   "license": "ISC",
   "dependencies": {
      "moment": "^2.19.1"
   },
   "dev-dependencies": {
      "webpack": "^3.8.1"
   }
}
```

Webpack use case

JS

- webpack index.js bundle.js

Webpack находит все выражения require из **index.js** и заменяет соответствующим кодом, чтобы получился единый выходной файл **bundle.js**.

Параметры можно сохранить в конфигурационный файл *webpack.config.js*

Bundler Approach

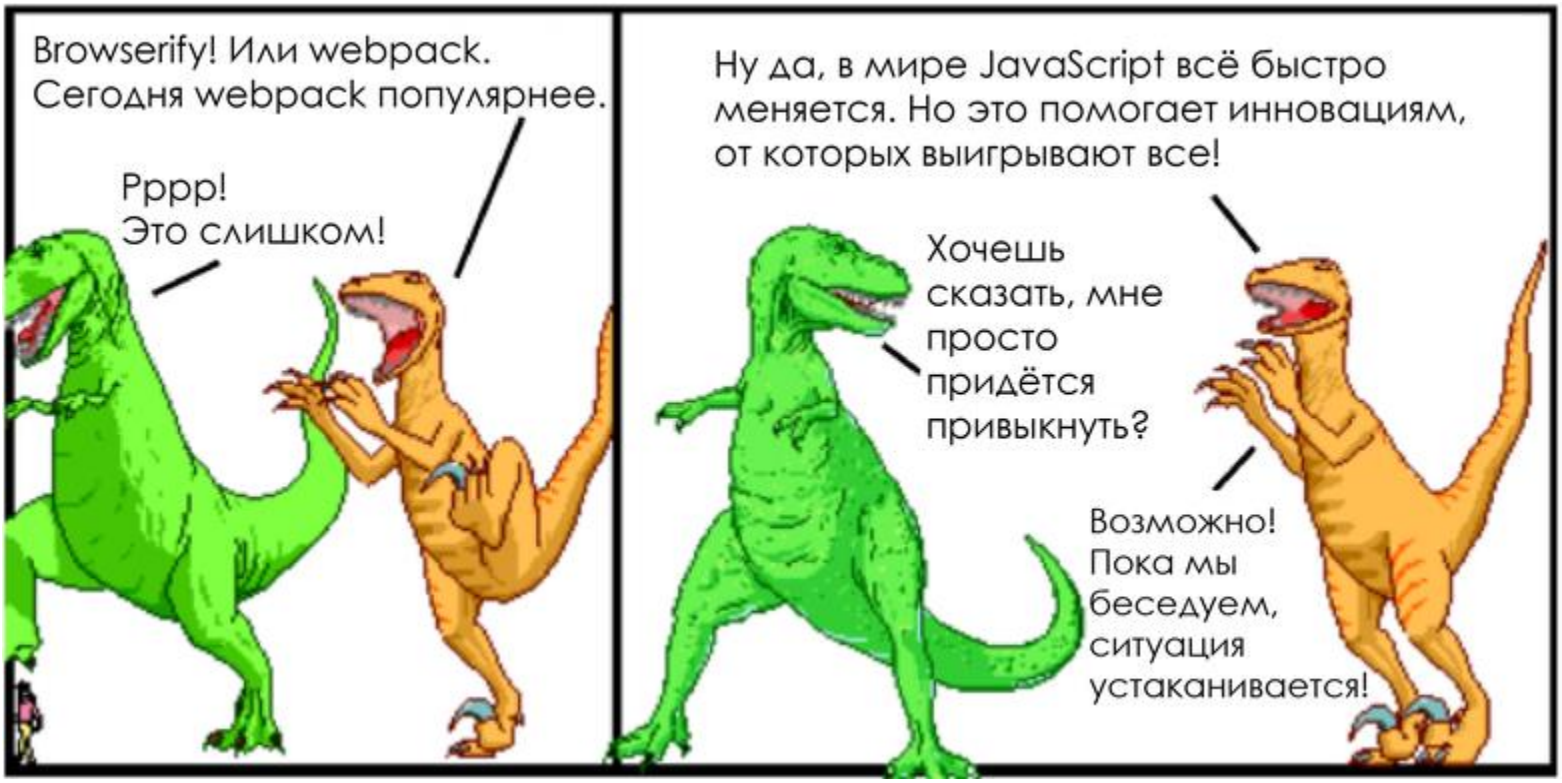
JS

- Создать html файл
- Используя командную строку через npm установить необходимые пакеты
  `npm install <package>`
- Собрать пакеты в bundle.js файл
- Подключить bundle.js файл
- Использовать функции из добавленного файла

Bundler Approach

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JavaScript Example</title>
  <script src="bundle.js"></script>
</head>
<body>
  <h1>Hello from HTML!</h1>
</body>
</html>
```

Bundler Approach



TypeScript and Babel

ТРАНСПИЛЯЦИЯ КОДА

Транспиляция кода

JS

- Транспилирование — конвертация кода с одного языка на другой язык.
- В браузерах медленно появляются новые features, поэтому создаются языки с экспериментальными возможностями, которые транспилируются в JavaScript.

Транспиляция кода

JS

- http://coffeescript.org/ – создан в 2010г. Опциональное использование скобок, значимые отступы (whitespace).
- https://babeljs.io/ - транспиляция JavaScript кода стандарта ES2015 и выше в широко поддерживаемый стандарт ES5.
- http://www.typescriptlang.org/ - типизированный JavaScript

CoffeeScript


```
# Assignment:
number = 42
opposite = true
# Conditions:
number = -42 if opposite
# Functions:
square = (x) -> x * x
# Objects:
math =
    root: Math.sqrt
    square: square
    cube: (x) -> x * square x

http://coffeescript.org/
```

```
var math, number, opposite, square
= [].slice;
number = 42; opposite = true;
if (opposite) {
    number = -42;
}
square = function(x) {
    return x * x;
};
math = {
    root: Math.sqrt,
    square: square,
    cube: function(x) {
        return x * square(x);
    }
};
```

Babel


```
var fibonacci = {
  [Symbol.iterator]: function*() {
    var pre = 0, cur = 1;
    for (;;) {
      var temp = pre;
      pre = cur;
      cur += temp;
      yield cur;
    }
  }
}
```

```
var fibonacci = _defineProperty({},
Symbol.iterator, regeneratorRuntime.mark(function
_callee() {
  var pre, cur, temp;
  return regeneratorRuntime.wrap(
function _callee$(_context) {
    while (1) {
      switch (_context.prev = _context.next) {
        case 0:
          pre = 0, cur = 1;
        case 1:
          temp = pre; pre = cur;  cur += temp;
          _context.next = 6;
          return cur;
        case 6:
          _context.next = 1;
          break;
        case 8:
        case "end":
          return _context.stop();
      }
    }
  }, _callee, this);
}));
```

TypeScript


```
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}

let greeter = new Greeter("world");

let button = document.createElement('button');
button.textContent = "Say Hello";
button.onclick = function() {
    alert(greeter.greet());
}

document.body.appendChild(button);
```

```
var Greeter = /** @class */
(function () {
function Greeter(message) {
this.greeting = message;
}
Greeter.prototype.greet = function
() {
return "Hello, " + this.greeting;
};
return Greeter;
}());
var greeter = new Greeter("world");
var button =
document.createElement('button');
button.textContent = "Say Hello";
button.onclick = function () {
alert(greeter.greet());
};
document.body.appendChild(button);
```

task runner and npm-скрипты

ИСПОЛЬЗОВАНИЕ СРЕДСТВА ЗАПУСКА ЗАДАЧ

Задачи task runners

JS

- Создание веб-сервера и автоматическая перезагрузка страницы в браузере при сохранении кода, слежение за изменениями в файлах проекта;
- Использование различных JavaScript, CSS и HTML препроцессоров (CoffeeScript, Less, Sass, Stylus, Jade и т.д.);
- Минификация CSS и JS кода, а также, оптимизация и конкатенация отдельных файлов проекта в один;
- Автоматическое создание вендорных префиксов (приставок к названию CSS свойства, которые добавляют производители браузеров для нестандартных свойств) для CSS.

Задачи task runners

JS

- Управление файлами и папками в рамках проекта - создание, удаление, переименование;
- Запуск и контроль выполнения внешних команд операционной системы;
- Работа с изображениями - сжатие, создание спрайтов, ресайз (png, jpg, svg и др.);
- Деплой (отправка на внешний сервер) проекта по FTP, SFTP, Git и т.д.
- Подключение и использование в проекте безгранично большого количества Node.js и Gulp утилит, программ и плагинов.

Task runners

JS

- Grunt – до 2013 г.
- Gulp – с 2013 г.
- NPM – встроенный scripting работает с первыми двумя

- Grunt и Gulp используют плагины, играющие роль обёртки для других инструментов, которым нужна командная строка.

Заключение

JS

- **диспетчер** пакетов для автоматической загрузки сторонних пакетов;
- **бандлер** для создания единых файлов скриптов;
- **транспилятор** для использования будущих возможностей JS;
- **средства запуска задач** для автоматизации разных операций в процессе сборки.

Не всё так плохо

JS

- Node экосистема - жизнеспособный вариант работы с фронтендом.
- Есть CLI для популярных frontedn фреймворков: ember-cli, angular-cli, create-react-app , vue-cli и т. д.

Источники

JS

По мотивам статьи Peter Jang

https://medium.com/@peterxjang/modern-javascript-explained-for-dinosaurs-f695e9747b70

И перевода от Mail.ru

https://habrahabr.ru/company/mailru/blog/340922

https://jakearchibald.com/2017/es-modules-in-browsers/