



# МОДУЛИ



ИЛЬЯ МЕДЖИДОВ / RAGEMARKET



# ИЛЬЯ МЕДЖИДОВ

CEO в RageMarket




[medzhidov@ragemarket.ru](mailto:medzhidov@ragemarket.ru)



# ПЛАН ЗАНЯТИЯ

1. Зачем нужны ES-модули?
2. Экспорт: export, export default
3. Импорт: import, смешанный импорт, import as, import \* as
4. Основные концепции webpack и его конфигурационный файл



# **ЗАЧЕМ НУЖНЫ ES- МОДУЛИ?**

# ЗАЧЕМ НУЖНЫ ES-МОДУЛИ?

- сложность JavaScript-приложений очень сильно выросла с 2006 г., когда появился jQuery: появилась сложная бизнес-логика на клиенте и необходимость управлять большим количеством данных, и как следствие — файлами различных типов
- если раньше было достаточно вручную при помощи тега `script` подключить несколько скриптов на HTML-страницу и легко следить за их зависимостями, то современное приложение при таком подходе невозможно поддерживать и расширять



# УСТАРЕВШИЙ ПОДХОД К МОДУЛЯМ

- паттерн проектирования «модуль» (может быть немедленно вызываемой функцией или функцией-конструктором)
- jQuery-плагины
- JavaScript в стиле ООП



# ПРОБЛЕМЫ ПРИ УСТАРЕВШЕМ ПОДХОДЕ

- конфликты имен модулей в глобальной области видимости
- необходимо вручную следить за последовательностью загрузки и инициализации модулей с учетом зависимостей между ними (для каждой HTML-страницы)

# ПОДХОДЫ К ОРГАНИЗАЦИИ МОДУЛЕЙ

- AMD (Asynchronous Module Definition) – одна из первых систем организации модулей
- CommonJS – система модулей, используемая Node.js

```
1  const students = require('./students.js');
2
3  console.log(students.getGroups());
4
5  // в students.js
6  exports.getGroups = () => {
7    // ...
8  };
```

- UMD (Universal Module Definition) – система модулей, совместимая с системой AMD и CommonJS.





## КАК ПРАВИЛЬНО РАЗБИВАТЬ КОД НА МОДУЛИ?

Модуль — это файл с кодом, из которого экспортируется хотя бы одна переменная (иначе его нельзя переиспользовать).

Один класс, компонент или библиотека — один модуль.



**ЭКСПОРТ: EXPORT,  
EXPORT DEFAULT**

# ИМЕНОВАННЫЙ ЭКСПОРТ

Ключевое слово `export` ставится перед объявлением переменных, функций и классов.

```
export const getStudentsByGroup = (students, group) =>
  students.filter(student => student.group === group);
```

В случае экспорта заранее объявленной переменной/функции/класса необходимо взять их название в фигурные скобки:

```
1  const getStudentsByGroup = (students, group) =>
2    students.filter(student => student.group === group);
3
4  export { getStudentsByGroup }; // необходимо взять в фигурные скобки
```

Можно экспортировать сразу несколько переменных:

```
export { getStudentsByGroup, getGroupsByCourse };
```

# ЭКСПОРТ ПО УМОЛЧАНИЮ

С использованием ключевого слова **default**, в одном файле может быть только один дефолтный экспорт:

```
const courseUtils = { ... };  
export default courseUtils;
```

Можно экспортировать по умолчанию и новый объект:

```
1 export default {  
2   getStudentsByGroup, // короткая запись ES6 для getStudentsByGroup: getStudentsByGroup  
3   getGroupsByCourse,  
4 };  
◀ ▶
```

## ПЕРЕИМЕНОВАНИЕ ПРИ ЭКСПОРТЕ (EXPORT AS)

При помощи ключевого слова **as** можно экспортировать переменную/функцию/класс под другим именем:

```
export { getStudentsByGroup as getStudents, getGroupsByCourse as getGroups };
```

---

**ИМПОРТ: IMPORT,  
СМЕШАННЫЙ ИМПОРТ,  
IMPORT AS, IMPORT \* AS**

# ИМЕНОВАННЫЙ ИМПОРТ

```
1 // модуль courseUtils .js
2 const courseUtils = {...};
3
4 export const getStudentsByGroup = (students, group) =>
5   students.filter(student => student.group === group);
6
7 export const getGroupsByCourse = (groups, course) =>
8   students.filter(group => group.course === course);
9
10 export default courseUtils;
```

Импорт одной или нескольких переменных, функций или классов по имени:

```
import { getStudentsByGroup, getGroupsByCourse } from './courseUtils.js';
```

Имя при экспорте и при импорте в данном случае должно совпадать.

# ESLINT

ESLint с настройками Airbnb будет "ругаться" на расширение модуля при импорте. Поэтому (если вы используете бандлер - Webpack), расширение указывать не нужно.

Но браузер (без бандлера) синтаксис без расширения (.js или .mjs) принимать не будет.

Настройка ESLint:

```
"rules": {  
  "import/extensions": [  
    "error",  
    "ignorePackages"  
  ]  
}
```

Детали [по ссылке](#).



---

# .MJS

Расширение `mjs` было предложено разработчиками Node.js для явного указания того, что файл использует систему ESM.

Браузеру и Webpack'у наличие расширения `mjs` не принципиально.

Ключевое - если вы используете модули без бандлера (нативно в браузере), веб-сервер должен отдавать для файлов `mjs` заголовок `Content-Type: application/javascript`

# ИМПОРТ ЗНАЧЕНИЯ ПО УМОЛЧАНИЮ

```
1 // модуль courseUtils.js
2 const courseUtils = {...};
3
4 export const getStudentsByGroup = (students, group) =>
5     students.filter(student => student.group === group);
6
7 export const getGroupsByCourse = (groups, course) =>
8     students.filter(group => group.course === course);
9
10 export default courseUtils;
```

```
import courseUtils from './courseUtils.js';
```

имя может быть и другим, не обязательно courseUtils

# СМЕШАННЫЙ ИМПОРТ

Импорт по умолчанию и по имени в одной строке

```
1 import courseUtils, { getStudentsByGroup, getGroupsByCourse } from './courseUtils.js';  
2 // вместо:  
3 import courseUtils from './courseUtils.js';  
4 import { getStudentsByGroup, getGroupsByCourse } from './courseUtils.js';
```

# ИМПОРТ ВСЕГО СОДЕРЖИМОГО МОДУЛЯ

`import * as`, при этом необходимо дать модулю имя

```
import * as utils from './courseUtils.js';
```

Обращаться к конкретным переменным через `utils`:

```
const { getStudentsByGroup, getGroupsByCourse, courseUtils } = utils;
```

## ИМПОРТ С ПЕРЕИМЕНОВАНИЕМ (IMPORT AS)

Если после импорта переменной хочется обращаться к ней по имени, отличающимся от того, с которым ее экспортировали:

```
import { getStudentsByGroup as getStudents, getGroupsByCourse as getGroups } from './courseUtils.js';
```

# КАК В ДРУГОМ МОДУЛЕ ИМПОРТИРОВАТЬ ОБЪЕКТ COURSEUTILS?

```
1  // модуль courseUtils.js
2  const courseUtils = {...};
3
4  export const sortStudentsByMark = students => (
5    students.sort((a, b) => {
6      if (a.mark > b.mark) {
7        return 1;
8      }
9      if (a.mark < b.mark) {
10       return -1;
11     }
12     return 0;
13   })
14 );
15
16 export const getBestStudent = students => sortStudentsByMark(students)[0];
17
18 export default courseUtils;
```

## ВАРИАНТЫ:

1. `import { courseUtils } from './courseUtils.js';`
2. `import courseUtils from './courseUtils.js';`
3. `import * from './courseUtils.js';`
4. `import { default as courseUtils } from './courseUtils.js';`

# КАК ИМПОРТИРОВАТЬ ФУНКЦИЮ SORTSTUDENTSBYMARK ПОД ИМЕНЕМ SORTSTUDENTS?

```
1  // модуль courseUtils.js
2  const courseUtils = {...};
3
4  export const sortStudentsByMark = students => (
5    students.sort((a, b) => {
6      if (a.mark > b.mark) {
7        return 1;
8      }
9      if (a.mark < b.mark) {
10       return -1;
11     }
12     return 0;
13   })
14 );
15
16 export const getBestStudent = students => sortStudentsByMark(students)[0];
17
18 export default courseUtils;
```



## ВАРИАНТЫ:

1. `import * as sortStudents from './courseUtils.js';`
2. `import sortStudentsByMark as sortStudents from './courseUtils.js';`
3. `import { sortStudentsByMark as sortStudents } from './courseUtils.js';`

# КАК ОБРАТИТЬСЯ К ФУНКЦИИ GETBESTSTUDENT?

```
1  // модуль courseUtils.js
2  const courseUtils = {...};
3
4  export const sortStudentsByMark = students => (
5    students.sort((a, b) => {
6      if (a.mark > b.mark) {
7        return 1;
8      }
9      if (a.mark < b.mark) {
10       return -1;
11      }
12     return 0;
13   })
14 );
15
16 export const getBestStudent = students => sortStudentsByMark(students)[0];
17
18 export default courseUtils;
```

```
import * as utils from './courseUtils.js';
```

## ВАРИАНТЫ:

1. `utils.getBestStudent()`
2. `getBestStudent()`
3. нельзя к ней обратиться

# КАК ИСПОЛЬЗОВАТЬ МОДУЛИ СЕЙЧАС?

Поддержка модулей присутствует в свежих версиях популярных браузеров:

```
1 <script type="module">
2   import { groups } from './jsCourse.js';
3   console.log(groups);
4 </script>
```

Или вот так можем сказать браузеру, что в подгружаемом скрипте используются модули:

```
<script type="module" src="./jsCourse.js"></script>
```

Для более полной поддержки браузерами рекомендуется использовать транспайлеры, такие как Babel, а также сборщики, например, Webpack.



# COMMONJS

# MODULE.EXPORTS

В CommonJS, если мы хотим сделать имя (функцию, переменную либо объект) доступным из нашего модуля, то:

```
1 module.exports = {  
2   variable: { ... },  
3   method: function() { ... },  
4 };
```

# REQUIRE

Если мы хотим использовать имя, экспортированное из другого модуля, в своём модуле, то:

```
1  const mod = require('<path_to_module>');  
2  // mod.variable - доступ к конкретному имени  
3  // mod.method() - доступ к конкретному имени
```




## ЧТО ИСПОЛЬЗОВАТЬ?

Придётся использовать обе, т.к. платформа Node.js поддерживает ES Modules в экспериментальном режиме.

Детали на странице <https://nodejs.org/api/esm.html>





# ОСНОВНЫЕ КОНЦЕПЦИИ WEBRACK И ЕГО КОНФИГУРАЦИОННЫЙ ФАЙЛ

# WEBPACK. ОСНОВНЫЕ КОНЦЕПЦИИ

Webpack — сборщик пакетов (бандлов).

- **Бандл** (Bundle)

Бандлы состоят из некоторого количества модулей и содержат итоговые версии исходных файлов, которые уже прошли компиляцию.

- **граф зависимостей** (Dependency graph)

Когда webpack обрабатывает JavaScript-приложение, он строит внутренний граф зависимостей, который сопоставляет каждый модуль приложения и генерирует один или несколько бандлов.

- **точка входа в приложение** (Entry point)

Точка входа указывает, какой модуль webpack должен использовать, чтобы начать строить свой граф зависимостей. webpack выясняет, от каких модулей и библиотек зависит эта точка входа (напрямую и через зависимости его зависимостей). По умолчанию точкой входа считается файл `./src/index.js`, но можно указать другой (или несколько других) в конфигурационном файле webpack.

`webpack.config.js`:

```
1 | module.exports = {  
2 |   entry: './path/to/my/entry/file.js',  
3 | }
```

# WEBPACK. ВЫХОДНОЙ ФАЙЛ (OUTPUT)

Свойство `output` указывает Webpack, куда выводить создаваемые им бандлы и как их назвать. Название по умолчанию для главного выходного файла `./dist/main.js`.

`webpack.config.js`:

```
1  const path = require('path'); // Node.js модуль для разрешения путей файлов
2
3  module.exports = {
4    entry: './src/index.js',
5    output: {
6      path: path.resolve(__dirname, 'dist'),
7      filename: 'app.bundle.js',
8    },
9  };
```

Свойства `output.filename` и `output.path` указывают имя бандла, и куда его сохранить.

# WEBPACK. ЗАГРУЗЧИКИ (LOADERS)

Два основных свойства для настройки загрузчика:

- **test** показывает, какие типы файлов должны быть обработаны Webpack
- **use** указывает, какой загрузчик должен использоваться для трансформации файлов указанного типа.

`webpack.config.js`:

```
1  const path = require('path');
2
3  module.exports = {
4    output: {
5      filename: 'my-first-webpack.bundle.js',
6    },
7    module: {
8      rules: [
9        { test: /\.txt$/, use: 'raw-loader' },
10      ],
11    },
12  };
```

Когда в каком-то JavaScript-файле встретится `import` файла `txt`, то будет использоваться загрузчик `raw-loader` для его обработки перед добавлением в бандл (`raw-loader` выдаст содержимое `.txt`-файла как строку).

# WEBPACK. ПЛАГИНЫ (PLUGINS)

Оптимизация бандлов, управление картинками, HTML-файлами и т.д.

webpack.config.js:

```
1  const HtmlWebpackPlugin = require('html-webpack-plugin'); // устанавливается через npm
2  const webpack = require('webpack'); // для получения доступа ко встроенным плагинам
3
4  module.exports = {
5    module: {
6      rules: [
7        { test: /\.txt$/, use: 'raw-loader' },
8      ],
9    },
10   plugins: [
11     new HtmlWebpackPlugin({ template: './src/index.html' }),
12   ],
13  };
```

Плагин html-webpack-plugin генерирует HTML-файл и автоматически вставляет в него ссылки на сгенерированные бандлы (чтобы не приходилось это делать вручную).

## WEBPACK. MODE (РЕЖИМ)

Можно включить встроенную оптимизацию Webpack для конкретного окружения: **development** (разработка), **production** (продакшн) или **none** (не установлен)

webpack.config.js:

```
1 | module.exports = {  
2 |   mode: 'production',  
3 | };
```

Можно также передавать в качестве флага в командной строке.

# WEBPACK. СОВМЕСТИМОСТЬ С БРАУЗЕРАМИ

Webpack поддерживает все браузеры, совместимые с ES5 (IE8 и ниже не поддерживаются).

Для поддержки более старых версий необходимо сначала загрузить полифиллы для Promise для использования в `import()` и `require.ensure()`.



## ЧЕМУ МЫ НАУЧИЛИСЬ

- узнали, зачем нужны модули ES и как правильно разбивать код на модули
- использовать экспорт: `export`, `export default`
- использовать импорт: `import`, смешанный импорт, `import as`, `import * as`
- использовать модули сейчас
- основным концепциям Webpack и его настройке через конфигурационный файл



# ССЫЛКИ НА ДОПОЛНИТЕЛЬНЫЕ МАТЕРИАЛЫ

1. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import>
2. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export>
3. <https://webpack.js.org/>



Спасибо за внимание!  
Время задавать вопросы 😊

**ИЛЬЯ МЕДЖИДОВ**

✉ [medzhidov@ragemarket.ru](mailto:medzhidov@ragemarket.ru)