

Модуль 7. Введение в асинхронный JS

Модуль 8. Введение в модули и Node.js

В модуле 7, 8 мы рассмотрим:

- ✓ Promise
- ✓ async/await
- ✓ Введение в Fetch API
- ✓ Что такое модули
- ✓ Импорт/экспорт
- ✓ Что такое Node.js
- ✓ Запуск скриптов
- ✓ Пакетный менеджер npm



Promises в JavaScript

Promise — это специальный объект, который представляет собой результат асинхронной операции.

Состояния: **pending** (ожидание), **fulfilled** (выполнено успешно), **rejected** (выполнено с ошибкой).



Почему нужны Promises?



Проблемы с колбэками:

- ✓ Callback Hell (Ад коллбеков): сложность иерархии вложенных функций.
- ✓ Запутанность и трудности с отладкой.
- ✓ Трудности с обработкой ошибок.

Преимущества Promises:

- ✓ Улучшают читаемость кода.
- ✓ Упрощают обработку ошибок.
- ✓ Обеспечивают гибкость при работе с асинхронными операциями.

Пример использования Promise для выполнения запроса к API по адресу <https://restcountries.com/v3.1/all> с использованием fetch



```
fetch("https://restcountries.com/v3.1/all")
  .then((response) => {
    if (!response.ok) {
      throw new Error("Network response was not ok");
    }
    return response.json();
  })
  .then((data) => {
    console.log(data); // Здесь вы можете обработать данные
  })
  .catch((error) => {
    console.error("There has been a problem with your fetch operation:", error);
  });
```

Вот как можно переписать этот пример с использованием async/await



```
async function fetchCountries() {
  try {
    const response = await fetch('https://restcountries.com/v3.1/all');
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    const data = await response.json();
    console.log(data); // Здесь вы можете обработать данные
  } catch (error) {
    console.error('There has been a problem with your fetch operation:', error);
  }
}

fetchCountries();
```

Что такое модули в JavaScript?



Модули в JavaScript позволяют организовать код в отдельные блоки, которые можно использовать повторно в других частях приложения или даже в других проектах. Это помогает структурировать код, улучшить его читаемость и упростить поддержку.

Основные концепции модулей включают:

- ✓ **Экспорт (export):** Позволяет определить, какие части модуля будут доступны для использования в других модулях.
- ✓ **Импорт (import):** Позволяет включать и использовать экспортированные части других модулей в текущем модуле.

Модули - атрибут type в HTML



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <script type="module" defer src="script.js"></script>
  </head>
  <body></body>
</html>
```


Помимо *index.html* и *script.js* создаем еще один файл *shoppingCart.js*



```
JS shoppingCart.js > ...  
1  // экспорт  
2  console.log("exp module");  
3  
4  const shippingCost = 11;  
5  const cart = [];
```

Подключаем в *script.js* файл *shoppingCart.js* с помощью директивы import



<> index.html X

JS script.js

X

JS shoppingCart.js

JS script.js

```
1  // импорт
2  import "../shoppingCart.js";
3
4  console.log("import module");
```

Создаем в модуле функцию и экспортируем



```
index.html  JS script.js  JS shoppingCart.js X

JS shoppingCart.js > ...
1  // экспорт
2  console.log("exp module");
3
4  const shippingCost = 11;
5  const cart = [];
6
7  // объявляем функцию для добавления товара в корзину и экспортируем
8  export const addToCart = function (product, quantity) {
9      cart.push({ product, quantity });
10     console.log(`${product} ${quantity} добавлено в корзину`);
11 };
```

Импортируем функцию и вызываем



<> index.html X

JS script.js X

JS shoppingCart.js

JS script.js

```
1  // импортируем функцию
2  import { addToCart } from "../shoppingCart.js";
3
4  console.log("import module");
5
6  // вызываем импортированную функцию
7  addToCart("Хлеб", 2);
```

Экспортируем несколько значений



<> index.html

JS script.js

JS shoppingCart.js X

JS shoppingCart.js > ...

```
1  // экспорт
2  console.log("exp module");
3
4  const shippingCost = 11;
5  const cart = [];
6
7  // объявляем функцию для добавления товара в корзину и экспортируем
8  export const addToCart = function (product, quantity) {
9      cart.push({ product, quantity });
10     console.log(`${product} ${quantity} добавлено в корзину`);
11 };
12
13 // экспортируем стоимость доставки и корзину
14 export { shippingCost, cart };
```

Импортируем несколько значений



<> index.html

JS script.js

×

JS shoppingCart.js

JS script.js

```
1  // импортируем функцию
2  import { addToCart, shippingCost, cart } from "../shoppingCart.js";
3
4  console.log("import module");
5
6  // вызываем импортированную функцию
7  addToCart("Хлеб", 2);
8  addToCart("Молоко", 4);
9
10 // смотрим на импортированные значения
11 console.log(shippingCost, cart);
```

Экспортируем все разом



<> index.html

JS script.js

JS shoppingCart.js X

JS shoppingCart.js > ...

```
1  const shippingCost = 11;
2  const cart = [];
3
4  // объявляем функцию для добавления товара в корзину
5  const addToCart = function (product, quantity) {
6      cart.push({ product, quantity });
7      console.log(`${product} ${quantity} добавлено в корзину`);
8  };
9
10 // экспортируем всё
11 export { shippingCost, cart, addToCart };
```

Импортируем все в один объект



<> index.html

JS script.js



JS shoppingCart.js

JS script.js

```
1  // импортируем все в один объект
2  import * as ShoppingCart from "./shoppingCart.js";
3
4  // вызываем импортированную функцию
5  ShoppingCart.addToCart("Хлеб", 2);
6  ShoppingCart.addToCart("Молоко", 4);
7
8  // смотрим на импортированные значения
9  console.log(ShoppingCart.shippingCost);
10 console.log(ShoppingCart.cart);
11 console.log(ShoppingCart.addToCart);
```



Default export



```
index.html JS script.js JS shoppingCart.js X
JS shoppingCart.js > ...
1  const shippingCost = 11;
2  const cart = [];
3
4  // экспортируем одно значение
5  export default function (product, quantity) {
6      cart.push({ product, quantity });
7      console.log(`${product} ${quantity} добавлено в корзину`);
8  }
```

Default export



 index.html JS script.js X JS shoppingCart.js

JS script.js

```
1  // импортируем дефолтное значение
2  import add from "./shoppingCart.js";
3
4  // вызываем импортированную функцию
5  add("Хлеб", 2);
6  add("Молоко", 4);
```

Top level await в модулях



```
index.html JS script.js X JS shoppingCart.js
JS script.js > ...
1  const response = await fetch("https://jsonplaceholder.
   typicode.com/posts");
2  const data = await response.json();
3  console.log(data);
4
5  console.log("hello");
```

npm (Node Package Manager)

Это менеджер пакетов для JavaScript и крупнейший репозиторий программного обеспечения в мире. Он используется для управления зависимостями в проектах, написанных на JavaScript, и предоставляет доступ к тысячам библиотек и инструментов, которые могут использоваться для ускорения разработки.

Основные функции npm:

- ✓ Управление пакетами.
- ✓ Создание и публикация пакетов.
- ✓ Управление зависимостями.



Инициализируем проект



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\МГУ\JS_level_2\Модуль 7,8\8> npm init
```

● Debugger attached.

```
Waiting for the debugger to disconnect...
```

```
Debugger attached.
```

```
This utility will walk you through creating a package.json file.
```

```
It only covers the most common items, and tries to guess sensible defaults.
```

```
See `npm help init` for definitive documentation on these fields  
and exactly what they do.
```

```
Use `npm install <pkg>` afterwards to install a package and  
save it as a dependency in the package.json file.
```

```
Press ^C at any time to quit.
```

```
package name: (8) my-pack
```

При этом создается конфигурационный файл `package.json`, где указаны все параметры



```
JS script.js  {} package.json X  <> index.html
8 > {} package.json > ...
1  {
2    "name": "my-pack",
3    "version": "1.0.0",
4    "main": "script.js",
5    "scripts": {
6      "test": "echo \"Error: no test specified\" && exit 1"
7    },
8    "author": "",
9    "license": "ISC",
10   "description": "",
11   "dependencies": { ...
13   }
14 }
```

С помощью команды `npm install` устанавливаем нужный пакет



PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

```
PS E:\МГУ\JS_level_2\Модуль 7,8\8> npm install lodash-es
```

- Debugger attached.

```
Waiting for the debugger to disconnect...
```

```
Debugger attached.
```

```
added 1 package, and audited 3 packages in 2s
```

```
found 0 vulnerabilities
```

```
Waiting for the debugger to disconnect...
```

- PS E:\МГУ\JS_level_2\Модуль 7,8\8> █

В проекте создается папка `node_modules` и в `package.json` прописывается зависимость



```
EXPL...  [Icons] JS script.js  package.json X

7
8
  node_modules
    leaflet
    lodash-es
  .package-lock.json
  index.html
  package-lock.json
  package.json
  script.js
  ~$Модуль7,8.pptx
  Модуль7,8.pptx

8 > package.json > ...
1  {
2    "name": "my-pack",
3    "version": "1.0.0",
4    "main": "script.js",
5    "scripts": {
6      "test": "echo \"Error: no test specified\" && exit 1"
7    },
8    "author": "",
9    "license": "ISC",
10   "description": "",
11   "dependencies": {
12     "leaflet": "^1.9.4",
13     "lodash-es": "^4.17.21"
14   }
15 }
```


Импортируем нужные файлы и используем



JS script.js X

8 > JS script.js

```
1  import sum from "../node_modules/lodash-es/sum.js";
2  import trim from "../node_modules/lodash-es/trim.js";
3
4  console.log(sum([1, 2, 3]));
5  console.log(trim("<p>hello</p>", "<p></p>()"));
6
```

Node.js



Node.js — это серверная платформа для исполнения **JavaScript**-кода вне браузера. Она позволяет создавать серверные приложения с использованием JavaScript, который изначально был создан для работы только в браузерах. Node.js построен на движке **V8** от Google, который используется в браузере Chrome для выполнения JavaScript.

Основные особенности Node.js



- ✓ **Асинхронная и событийно-ориентированная архитектура:** Node.js поддерживает неблокирующую модель ввода-вывода (I/O), что позволяет обрабатывать множество запросов одновременно без необходимости блокировать выполнение задач
- ✓ **Однопоточный:** Node.js работает в одном потоке, что снижает накладные расходы на управление потоками. Однако, благодаря асинхронности, он может эффективно обрабатывать множество соединений.
- ✓ **npm (Node Package Manager):** В комплекте с Node.js поставляется npm — менеджер пакетов, который облегчает управление библиотеками и модулями для Node.js. npm имеет огромный репозиторий модулей, которые можно легко установить и использовать в проектах.
- ✓ **Широкие возможности для создания серверов:** Node.js часто используется для создания веб-серверов, API, микросервисов и других серверных приложений.

Работа с Node.JS в командной строке



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

● PS E:\МГУ\JS_level_2\Модуль 7,8\node.js> node -v
v20.16.0
● PS E:\МГУ\JS_level_2\Модуль 7,8\node.js> node
Welcome to Node.js v20.16.0.
Type ".help" for more information.
> const name = "Denis"
undefined
> name
'Denis'
> let age = 22;
undefined
> age++
22
> age
23
> .exit
○ PS E:\МГУ\JS_level_2\Модуль 7,8\node.js> 
```

Первый скрипт на Node.JS



JS index.js X

JS index.js > ...

```
1  const fName = "Петр";
2  const lName = "Сидоров";
3  const message = `Привет, ${fName} ${lName}!!!`;
4
5  console.log(message);
6
```


PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

 powershell

```
PS E:\МГУ\JS_level_2\Модуль 7,8\node.js> node index.js
```

```
Привет, Петр Сидоров!!!
```

```
PS E:\МГУ\JS_level_2\Модуль 7,8\node.js> 
```

Чтение содержимого файла



JS index.js X

JS index.js > ...

```
1  // подключаем модуль для работы с файлами
2  const fs = require("fs");
3
4  // считываем содержимое файла
5  const textStart = fs.readFileSync("./txt/start.txt", "utf-8");
6  console.log(textStart);
7
```


PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

 powershell

- PS E:\МГУ\JS_level_2\Модуль 7,8\node.js> node index.js
Однажды, в студеную зимнюю пору
- PS E:\МГУ\JS_level_2\Модуль 7,8\node.js>

Запись в файл



JS index.js X

JS index.js > ...

```
1  // подключаем модуль для работы с файлами
2  const fs = require("fs");
3
4  // считываем содержимое файла
5  const textStart = fs.readFileSync("./txt/start.txt", "utf-8");
6
7  // запись в файл
8  const textOut = "Николай Некрасов. 1861 г.";
9  fs.writeFileSync("./txt/output.txt", textOut);
10
```


PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

 powershell

```
● PS E:\МГУ\JS_level_2\Модуль 7,8\node.js> node index.js
○ PS E:\МГУ\JS_level_2\Модуль 7,8\node.js> 
```

Чтение из нескольких файлов и запись в файл



JS index.js X

JS index.js > ...

```
1  // подключаем модуль для работы с файлами
2  const fs = require("fs");
3
4  // считываем содержимое файла
5  const textStart = fs.readFileSync("./txt/start.txt", "utf-8");
6  const textMiddle = fs.readFileSync("./txt/middle.txt", "utf-8");
7  const textEnd = fs.readFileSync("./txt/end.txt", "utf-8");
8  const author = "Николай Некрасов. 1861 г.";
9
10 const output = `${textStart}\n${textMiddle}\n${textEnd}\n${author}`;
11
12 // запись в файл
13 fs.writeFileSync("./txt/output.txt", output);
```


Асинхронный вариант чтения и записи



JS index.js X

JS index.js > ...

```
1  // подключаем модуль для работы с файлами
2  const fs = require("fs");
3
4  // асинхронный способ работы с файлами
5  fs.readFile("./txt/start.txt", "utf-8", function (err1, data1) {
6      fs.readFile("./txt/middle.txt", "utf-8", function (err2, data2) {
7          fs.readFile("./txt/end.txt", "utf-8", function (err3, data3) {
8              const author = "Николай Некрасов. 1861 г.";
9              const output = `${data1}\n${data2}\n${data3}\n${author}`;
10             fs.writeFile("./txt/output.txt", output, "utf-8", function (err4) {
11                 console.log("Файл записан успешно");
12             });
13         });
14     });
15 });
```

Запуск сервера node.js



JS index.js X

JS index.js > ...

```
1  const fs = require("fs");
2  const http = require("http");
3  const url = require("url");
4
5
6  const server = http.createServer(function (request, response) {
7    | response.end("Server started");
8  });
9
10 server.listen(8080, "localhost", function () {
11   | console.log("Listening port 8080");
12 });
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

```
PS E:\МГУ\JS_level_2\Модуль 7,8\node.js> node index.js
Listening port 8080
```

Настройка роутинга



JS index.js X

JS index.js > [?] server > http.createServer() callback

```
1  const fs = require("fs");
2  const http = require("http");
3
4  const server = http.createServer(function (request, response) {
5    const pathName = request.url;
6
7    if (pathName === "/" || pathName === "/ovweriew") {
8      response.end("ovweriew");
9    } else if (pathName === "/product") {
10     response.end("product");
11   } else {
12     response.writeHead(404, { "Content-type": "text/html" });
13     response.end("<h1>Page not found</h1>");
14   }
15 });
16
17 server.listen(8080, "localhost", function () {
18   console.log("Listening port 8080");
19 });
```

Подключение HTML-шаблонов



JS index.js X

JS index.js > ...

```
1  const fs = require("fs");
2  const http = require("http");
3
4  const tempOverview = fs.readFileSync("./templates/overview.html", "utf-8");
5  const tempProduct = fs.readFileSync("./templates/product.html", "utf-8");
6
7  const server = http.createServer(function (request, response) {
8    const pathName = request.url;
9
10   if (pathName === "/" || pathName === "/ovweriew") {
11     response.writeHead(200, { "Content-type": "text/html" });
12     response.end(tempOverview);
13   } else if (pathName === "/product") {
14     response.writeHead(200, { "Content-type": "text/html" });
15     response.end(tempProduct);
16   } else {
17     response.writeHead(404, { "Content-type": "text/html" });
18     response.end("<h1>Page not found</h1>");
19   }
20 });
21
22 server.listen(8080, "localhost", function () {
23   console.log("Listening port 8080");
24 });
```

Спасибо
за внимание!
Ваши вопросы...



Учебный центр «СПЕЦИАЛИСТ» – Ваш путь к успеху



info@specialist.ru



+7 (495) 232-32-16

