Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Node или **Node.js** — это серверная платформа, основанная на движке V8, компилирующая JavaScript код в машинный код.

Node.js использует **событийно-ориентированную** модель и **неблокирующую ввод/вывод** архитектуру, что делает его «легковесным» и эффективным. Это **не фреймворк**, и **не библиотека**, это **среда выполнения** JavaScript.

Node.js использует неблокирующие ввод/вывод операции. При множественных запросах:

- главный поток не блокируется операциями ввода/вывода;
- сервер продолжает обслуживать запросы.
- используется **асинхронный код**.

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Установка Node.js:

-с официального сайта: *https://nodejs.org/en/download/*;

-с использованием менеджера пакетов. Например, *brew install node*

Популярным менеджером версий Node.js является **nvm**. Это средство позволяет удобно переключаться между различными версиями Node.js, с его помощью можно, например, установить и попробовать новую версию Node.js, после чего, при необходимости, вернуться на старую. Nvm пригодится и в ситуации, когда нужно испытать какой-нибудь код на старой версии Node.js.

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Установка Node.js:

-с официального сайта: *https://nodejs.org/en/download/*;

-с использованием менеджера пакетов. Например, *brew install node*

Популярным менеджером версий Node.js является **nvm**. Это средство позволяет удобно переключаться между различными версиями Node.js, с его помощью можно, например, установить и попробовать новую версию Node.js, после чего, при необходимости, вернуться на старую. Nvm пригодится и в ситуации, когда нужно испытать какой-нибудь код на старой версии Node.js.

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Пример создания сервера. Создайте файл с именем app.js, содержащий следующий код:

```
1   const http = require('http')
2   const hostname = '127.0.0.1'
3   const port = 80
4   const server = http.createServer((req, res) => {
5   res.statusCode = 200
6   res.setHeader('Content-Type', 'text/plain')
7   res.end('Hello World\n')
8   })
9   server.listen(port, hostname, () => {
10  console.log(`Server running at http://${hostname}:${port}/`)
11  })
```

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Запустите веб-сервер, используя команду: `node app.js`

Для проверки сервера откройте какой-нибудь браузер и введите в адресной строке http://127.0.0.1:3000 , то есть — тот адрес сервера, который будет выведен в консоли после его успешного запуска. Если всё работает как надо — на странице будет выведено «Hello World».

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Разберём этот пример:

- в начале, производится подключение **модуля http.**
платформа Node.js является обладателем замечательного стандартного набора модулей, в который входят отлично проработанные механизмы для работы с сетью.
- метод createServer() объекта http создаёт новый HTTP-сервер и возвращает его.
Сервер настроен на прослушивание определённого порта на заданном хосте. Когда сервер будет готов, вызывается соответствующий коллбэк, сообщающий нам о том, что сервер работает.

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Когда сервер получает запрос, вызывается **событие request**, предоставляющее два объекта. Первый — это запрос **(req , объект http.IncomingMessage)**, второй — ответ **(res, объект http.ServerResponse).**
Они представляют собой важнейшие механизмы обработки HTTP-запросов.

Первый предоставляет в наше распоряжение сведения о запросе. В нашем простом примере этими данными мы не пользуемся, но, при необходимости, с помощью объекта req можно получить доступ к заголовкам запроса и к переданным в нём данным.

Второй нужен для формирования и отправки ответа на запрос.

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

В данном случае ответ на запрос мы формируем следующим образом. Сначала устанавливаем свойство statusCode в значение 200, что указывает на успешное выполнение операции:

res.statusCode = 200

Далее, мы устанавливаем заголовок Content-Type:

res.setHeader('Content-Type', 'text/plain')

После этого мы завершаем подготовку ответа, добавляя его содержимое в качестве аргумента метода end() :

res.end('Hello World\n')

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Фреймворки и вспомогательные инструменты для Node.js

Node.js — это низкоуровневая платформа. Для того чтобы упростить разработку для неё и облегчить жизнь программистам, было создано огромное количество **библиотек**:

- **Express**. Эта библиотека предоставляет разработчику предельно простой, но мощный инструмент для создания веб-серверов. Ключом к успеху Express стал минималистический подход и ориентация на базовые серверные механизмы без попытки навязать некое видение «единственно правильной» серверной архитектуры.

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Meteor. Это — мощный фулстек-фреймворк, реализующий изоморфный подход к разработке приложений на JavaScript и к использованию кода и на клиенте, и на сервере. Когда-то Meteor представлял собой самостоятельный инструмент, включающий в себя всё, что только может понадобиться разработчику. Теперь он, кроме того, интегрирован с фронтенд-библиотеками, такими, как React , Vue и Angular . Meteor, помимо разработки обычных веб-приложений, можно использовать и в мобильной разработке.

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Koa. Этот веб-фреймворк создан той же командой, которая занимается работой над Express. При его разработке, в основу которой легли годы опыта работы над Express, внимание уделялось простоте решения и его компактности. Этот проект появился как решение задачи внесения в Express серьёзных изменений, несовместимых с другими механизмами фреймворка, которые могли бы расколоть сообщество.

Next.js. Этот фреймворк предназначен для организации серверного рендеринга React -приложений.

Micro. Это — весьма компактная библиотека для создания асинхронных HTTP-микросервисов.

Socket.io. Это библиотека для разработки сетевых приложений реального времени.

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Различия между платформой Node.js и браузером

Окружение

В браузере основной объём работы приходится на выполнение различных операций с веб-документами посредством DOM, а также — на использование других API веб-платформы, таких, скажем, как механизмы для работы с куки-файлами. Всего этого в Node.js, конечно, нет. Тут нет ни объекта document, ни объекта window, равно как и других объектов, предоставляемых браузером.

В браузере, в свою очередь, нет тех программных механизмов, которые имеются в среде Node.js и существуют в виде модулей, которые можно подключать к приложению. Например, это API для доступа к файловой системе.

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

В среде Node.js разработчик полностью контролирует окружение.

- вы точно знаете, например, на какой версии Node.js будет работать ваш проект.
- вы можете, не опасаясь проблем, пользоваться новейшими возможностями языка.

Так как JavaScript крайне быстро развивается, браузеры просто не успевают достаточно оперативно реализовать все его новшества. К тому же, далеко не все пользователи работают на самых свежих версиях браузеров.

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Подключение внешних модулей:

- в Node.js используется система модулей **CommonJS**,
- в браузерах - стандарт **ES Modules**.

- в Node.js, для подключения внешнего кода, используется конструкция **require(),** а в браузерном коде — **import**.

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Выход из Node.js-приложения

При выполнении программы в **консоли**, завершить её работу можно, воспользовавшись со**четанием клавиш ctrl+c .**

Программные способы завершения работы приложений:

Модуль ядра process предоставляет удобный метод, который позволяет осуществить программный выход из Node.js-приложения. Выглядит это так: **process.exit()**

Когда Node.js встречает в коде такую команду, это приводит к тому, что его процесс **мгновенно завершается**.

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Выход из Node.js-приложения

Надо отметить, что работа программы самостоятельно завершится естественным образом после того, как она выполнит все заданные в ней действия. Однако в случае с Node.js часто встречаются программы, которые, в идеальных условиях, рассчитаны на работу неопределённой длительности.

Для завершения работы подобных программ нужно воспользоваться сигналом **SIGTERM** и выполнить необходимые действия с помощью соответствующего **обработчика**.

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Сигналы — это средства взаимодействия процессов в стандарте POSIX (Portable Operating System Interface). Они представляют собой уведомления, отправляемые процессу для того, чтобы сообщить ему о неких событиях.

Например, сигнал **SIGKILL** сообщает процессу о том, что ему нужно немедленно завершить работу. Он работает так же, как **process.exit().**

Сигнал **SIGTERM** сообщает процессу о том, что ему нужно осуществить процедуру нормального завершения работы.

Отправить такой сигнал можно и из самой программы, воспользовавшись следующей командой:

process.kill(process.pid, 'SIGTERM')

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Использование Node.js в режиме REPL

Аббревиатура REPL расшифровывается как Read-Evaluate-Print-Loop (цикл «чтение — вычисление — вывод»). Использование REPL — это отличный способ быстрого исследования возможностей Node.js.

Как вы уже знаете, для **запуска скриптов** в Node.js используется команда node , выглядит это так:

node script.js

Если ввести такую же команду, но не указывать имя файла, Node.js будет запущен в режиме REPL:

node

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Использование Node.js в режиме REPL

Node.js теперь находится в режиме ожидания. Система ждёт, что мы введём в командной строке какой-нибудь JavaScript-код, который она будет выполнять.

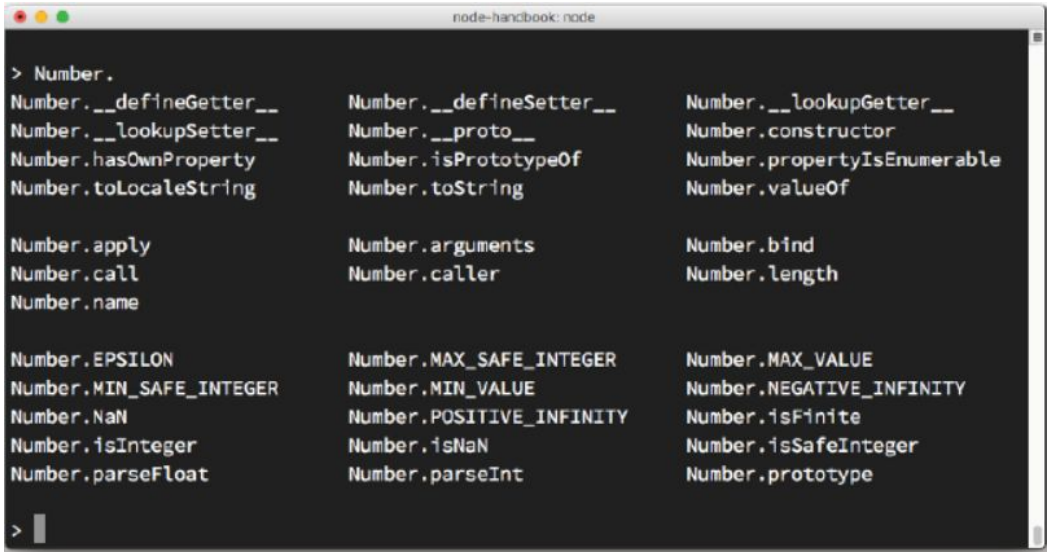
Автозавершение команд с помощью клавиши Tab

REPL — это интерактивная среда. Если в процессе написания кода нажать клавишу Tab на клавиатуре, REPL попытается автоматически завершить ввод, подобрав, например, подходящее имя уже объявленной вами переменной или имя некоего стандартного объекта.

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Использование Node.js в режиме REPL

Введите в командную строку имя какого-нибудь стандартного объекта JavaScript, например — Number, добавьте после него точку и нажмите клавишу Tab . REPL выведет список свойств и методов объекта, с которыми может взаимодействовать разработчик:



Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Использование Node.js в режиме REPL

Команды, начинающиеся с точки

В режиме REPL можно пользоваться некоторыми специальными командами, которые начинаются с точки. Вот они:

- Команда .help выводит справочные сведения по командам, начинающимся с точки.
- Команда .editor переводит систему в режим редактора, что упрощает ввод многострочного JavaScript-кода. После того, как находясь в этом режиме, вы введёте всё, что хотели, для запуска кода воспользуйтесь командой Ctrl+D .

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Использование Node.js в режиме REPL

- Команда .break позволяет прервать ввод многострочного выражения. Её использование аналогично применению сочетания клавиш Ctrl+C .
- Команда .clear очищает контекст REPL, а так же прерывает ввод многострочного выражения.
- Команда .load загружает в текущий сеанс код из JavaScript-файла.
- Команда .save сохраняет в файл всё, что было введено во время REPL-сеанса.

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Использование Node.js в режиме REPL

- Команда .exit позволяет выйти из сеанса REPL, она действует так же, как два последовательных нажатия сочетания клавиш Ctrl+C. Надо отметить, что REPL распознаёт ввод многострочных выражений и без использования команды .editor .

Специальная переменная _

Переменная _ (знак подчёркивания) хранит результат последней выполненной операции. Эту переменную можно использовать в составе команд, вводимых в консоль.

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Использование Node.js в режиме REPL

Работа с аргументами командной строки в Node.js-скриптах

При запуске Node.js-скриптов им можно передавать аргументы. Вот обычный вызов скрипта:

node app.js

Передаваемые скрипту аргументы могут представлять собой как самостоятельные значения, так и конструкции вида ключ-значение. В первом случае запуск скрипта выглядит так:

node app.js flavio

Во втором — так:

node app.js name=flavio

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Использование Node.js в режиме REPL

Для того, чтобы получить ***доступ к аргументам командной строки***, используется стандартный объект Node.js ***process***. У него есть ***свойство argv***, которое представляет собой массив, содержащий, кроме прочего, ***аргументы, переданные скрипту при запуск***е.

Первый элемент массива argv содержит полный ***путь к файлу***, который выполняется при вводе команды node в командной строке.

Второй элемент — это путь к выполняемому файлу скрипта. Все остальные элементы массива, начиная с третьего, содержат то, что было передано скрипту при его запуске.

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Использование Node.js в режиме REPL

Вывод данных в консоль с использованием модуля console

Стандартный модуль Node.js ***console*** даёт разработчику массу ***возможностей по взаимодействию с командной строкой*** во время выполнения программы. В целом, это — то же самое, что объект console , используемый в браузерном JavaScript. Пожалуй, самый простой и самый широко используемый метод модуля console — это ***console.log()***, который применяется для ***вывода*** передаваемых ему строковых ***данных в консоль***:

```
const x = 'x'
const y = 'y'
console.log(x, y)
```

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Система модулей Node.js, использование команды exports

В Node.js имеется встроенная система модулей, каждый файл при этом считается самостоятельным модулем. Общедоступный функционал модуля, с помощью команды require, могут использовать другие модули:

```
const library = require('./library')
```

Здесь показан импорт модуля library.js, файл которого расположен в той же папке, в которой находится файл, импортирующий его.

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Система модулей Node.js, использование команды exports

Модуль, прежде чем будет смысл его импортировать, **должен** что-то **экспортировать**, **сделать общедоступным**. *Ко всему, что явным образом не экспортируется модулем, нет доступа извне.*

Собственно говоря, API **module.exports** позволяет **организовать экспорт** того, что будет **доступно внешним** по отношению к модулю **механизмам**.

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Система модулей Node.js, использование команды exports

Экспорт можно организовать двумя способами.

Первый заключается в *записи объекта в module.exports*, который является стандартным объектом, предоставляемым системой модулей. Это приводит к экспорту только соответствующего объекта:

```
const car = {
brand: 'Ford',
model: 'Fiesta'
}
module.exports = car
//..в другом файле
const car = require('./car')
```

Лекция № 9. Серверный и браузерный контексты исполнения скрипта. Node.js

Система модулей Node.js, использование команды exports

Второй способ заключается в том, что ***экспортируемый объект записывают в свойство объекта exports***. Такой подход позволяет экспортировать из модуля несколько объектов, и, в том числе — функций:

```
const car = {
brand: 'Ford',
model: 'Fiesta'
}
exports.car = car
```

В другом файле воспользоваться тем, что экспортировал модуль, можно так:

```
const items = require('./items')
items.car
```