ОНЛАЙН-ОБРАЗОВАНИЕ

OTUS

Самый простой back-end на NodeJS

Евгений Аристов

Дворжецкий Юрий

Меня хорошо слышно && видно?

Напишите в чат, если есть проблемы!

Ставьте + если все хорошо
Ставьте - если есть проблемы

Аушев Мустафа

- Backend разработчик
- Пишу на node.js/python/golang
- Архитектор (иногда)

Что сможете после вебинара

- Цель вебинара – написать самый простой back-end
- Мы долго проводили анализ и выбрали JS ☺️
- JS взрывает мозг, но на этом построено обучение ☺️
- Поэтому, если возникает вопрос – сразу задавайте)

Поехали!

01

Типы данных

Немного о JavaScript

- EcmaScript (ES) – стандарт, притом разных версий
- JavaScript - это, по сути, реализация
- Язык может исполняться в разных местах (в браузере, в NodeJS)
- То, с чего мы начнём сегодня - это ES 5.1 в браузере
- Это останется верным, когда пройдём ES6

Где запускаем JS

- В браузере есть консоль
- Ctrl + Shift + J - консоль
- Всегда можно что-то попробовать
- console.log(....)

Типы данных в JS

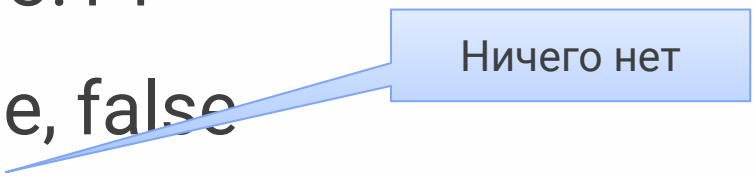
- В JS есть типы!
- Из 6 в ES5, в ES6 - их 7 + Symbol
- А вот переменные типа не имеют (динамическая типизация)
- Расширения (TypeScript, Flow) позволяют задать тип во время компиляции

Типы данных в JS

- String: "Привет", "
- Number: -1, 3.14
- Boolean: true, false
- Null: null
- Undefined: undefined
- Object: {x:1}, [0,1], function(){}, x => x + 5

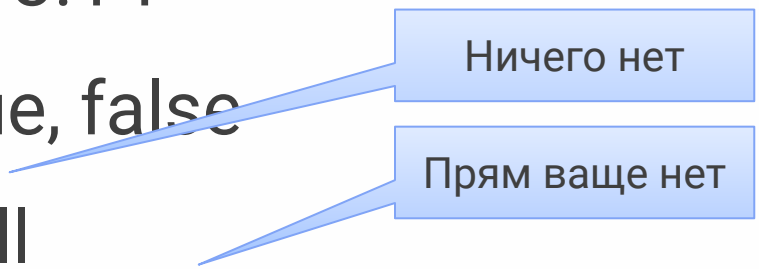
Типы данных в JS

- String: "Привет", ''
- Number: -1, 3.14
- Boolean: true, false
- Null: null
- Undefined: undefined
- Object: {x:1}, [0,1], function(){}, x => x + 5



Типы данных в JS

- String: "Привет", ''
- Number: -1, 3.14
- Boolean: true, false
- Null: null
- Undefined: undefined
- Object: {x:1}, [0,1], function(){}, x => x + 5



Типы данных в JS

- String: "Привет", ''
- Number: -1, 3.14
- Boolean: true, false
- Null: null
- Undefined: undefined
- Object: {x:1}, [0,1], function(){}, x => x + 5

Ничего нет

Прям ваще нет

Объект

Массив

Функция

Стрелочная функция

- Можно писать в одинарных кавычках, а можно в двойных
- Строчки юникодовские
- У строки есть длина
- Строки можно складывать
- Со строчками можно делать много всего
- Но они immutable: любая операция создает новую.

Упражнение

OTUS

```
// Чему равно?

('hello' + ' ' + "всем").length
```

Упражнение

```
// Чему равно?

('hello' + ' ' + "всем").length
10
```

Number

- Все числа - с плавающей запятой (double, 64 бит)
- NaN, Infinity, -Infinity – тоже числа
- Все операции тоже с плавающей запятой
- Деньги в них считать опасно

Упражнение

OTUS

```
7 / 2


1 / 0


Infinity - Infinity


-NaN
```

Упражнение

```
7 / 2
3.5

1 / 0
Infinity

Infinity - Infinity
NaN

-NaN
NaN
```

Разминка

OTUS

```
// Что будет выведено на экран?

console.log(0.25 + 0.5 === 0.75);



console.log(0.1 + 0.2 === 0.3);
```

Разминка

OTUS

```
// Что будет выведено на экран?

console.log(0.25 + 0.5 === 0.75);
true



console.log(0.1 + 0.2 === 0.3);
false
```

Null и Undefined

OTUS

```
var a;                          // undefined
console.log(a == null); // true

var a = null;                   // null
console.log(a == null); // true
```

- Объекты, массивы, функции - всё объект
- Да, функции могут иметь поля
- Объекты - ассоциативный массив
- Ключи - строчки
- Массивы - тоже ассоциативный массив

Null и Object

```
var obj = {};

var obj2 = { field: 100 };

var obj3 = {
    field1: 100,
    method: function () {
        this.field1 = 101;
    }
};
```

Упражнение

OTUS

```
var obj = {
    field: 100,
    method: function () {
        this.field = 101;
    }
};

console.log(obj.field);      // ?
console.log(obj['field']);   // ?

obj.method();
```

Массив

- На самом деле, тот же объект
- И также индексируется строчками ☺️
- Но есть ещё и длина и специфичные методы

Упражнение

OTUS

```
var arr = [0, 1, 2];

console.log(arr[1]);       // ?

arr.push(3);               // методов тьма

console.log(arr['1']);     // ?
```

```
function f1() { alert('!'); }

var f2 = function () { alert('!'); }

var f3 = () => { alert('!'); }  // ES6

f1(); // ?
f2(); // ?
f3(); // ?
```

```
var v = 200;
f();
function f() {
    alert(s);
    alert(v);
    var s = 100;
}

// Что будет
выведено?
```

Глобальный скоуп

```
var v = 200; // определяется в глобальном
объекте
console.log(v); // выведет v;
console.log(window.v); // выведет v;

window.document === document
```

Ваши вопросы?

02

Операторы

Операторы сравнения

```
10 == '10'              // ?

10 === '10'             // ?

null == undefined       // ?

false == 'false'        // ?
```

Операторы сравнения

OTUS

```
10 == '10'              // true

10 === '10'             // false

null == undefined       // true

false == 'false'        // false
```

Операторы сравнения

- === - рекомендуется
- == - на крайняк - никогда
- Алгоритм приведения типов в == не совсем совпадает с алгоритмом приведения типов
- На MDN ознакомьтесь

Оператор &&

OTUS

```
10 == '10' && flag



var obj = null;
var obj = {};
var obj = {field: false};
var obj = {field: {subfield: undefined}};
var obj = {field: {subfield: null}};
var obj = {field: {subfield: 'aaaa'}};

console.log(obj && obj.field && obj.field.subfield); // ?
```

```
function printTenNumbers() {
    for (var i = 0; i < 10; ++i) {
        console.log(i);
    }
    return 'OK';
}

printTenNumbers();

// Что будет выведено?
```

```
if (a < 10) {
    // do something
} else {
    return;
}
```

try-catch-finally

```
function f() {
    try {
        // do something dangerous
        throw 'error';
        throw new Error('!!!');
        return 0;
    } catch (err) {
        return 1;
    } finally {
        return 2;
    }
    return 3;
}
// что будет возвращено?
```

try-catch-finally

```
function f() {
    try {
        // do something dangerous
        throw 'error';
        throw new Error('!!!');
        return 0;
    } catch (err) {
        return 1;
    } finally {
        return 2;
    }
    return 3;
}
// что будет возвращено?
```

Ваши вопросы?

03

ООП

- ООП есть, но классов нет
- Классы нет и в ES6, хотя слово class есть
- И это только синтаксический сахар, классы - это функции
- Зато есть объекты
- Поля и методы задаются у объектов

Упражнение

OTUS

```
var obj = {
    x: 1,
    y: 2,
    sum: function() {
        return this.x + this.y;
    }
};

console.log(obj.sum()); // ?
```

Упражнение

OTUS

```
var obj = {
    x: 1,
    y: 2,
    sum: function() {
        return this.x + this.y;
    }
};
var fun = function() { return this.x + 3};
obj.fun2 = fun;

console.log(obj.sum() + obj.fun2()); // ?
```

- this всегда вычисляется в момент выполнения
- Можно писать this.несуществующееПоле
- Его можно менять, и он не всегда будет указывать на нужный объект
- Без this к полям/методам доступ не получить

Упражнение

OTUS

```
var fun1 = function(x) { this.x = x}

var obj = {};

fun1.call(obj, 5);

console.log(obj.x); // ?
```

Упражнение

OTUS

```
var fun1 = function(x) { this.x = x}

var obj = {};

fun1.call(obj, 5);

console.log(obj.x); // ?
```

Упражнение

OTUS

```
var fun1 = function(x) { this.x = x}

var obj = {};

var fun2 = fun1.bind(obj);

fun2(5);

console.log(obj.x); // ?
```

Упражнение

```
var fun1 = function(x) { this.x = x}

var obj = {};

var fun2 = fun1.bind(obj, 5);

fun2();

console.log(obj.x); // ?
```

Упражнение

OTUS

```
// Что это означает?

var func = function() {
    this.x = 0;
    this.y = 0;
}

var point = new func();
```

Упражнение

OTUS

```
// Что это означает?

var func = function(x, y) { this.x = x; this.y = y;}
var point = new func(1, 2);

console.log(func.x);
```

Синтаксический сахар ES6

OTUS

```
class Point {
    constructor() {
        this.x = 0;
        this.y = 0;
    }
}

var point = new Point();
```

Синтаксический сахар (ES6)

OTUS

- Только сахар
- Если браузер и поддерживает class, то он просто эмулирует это с выражениями с функциями
- Но очень похож на классы
- Поэтому будем считать, что классы есть

Упражнение

OTUS

```
class Point {

    constructor() {
        this.x = 0;
        this.y = 0;
    }

    length() {
        return Math.sqrt(this.x * this.x + this.y * this.y);
    }
}

console.log(new Point().length()); // ?
```

А что если написать все это не используя слово class

Синтаксический сахар ES6

OTUS

```
class Parent {
    constructor() {
        this.x = 0;
    }
}

class Child extends Parent {
    constructor() {
        super();
        this.y = 0;
    }
}
```

Ваши вопросы?

04

NodeJS

- NodeJS – не «серверный» JS

- Среда выполнения JS
- Есть встроенный package manager npm (его репозиторий доступен https://npmjs.com)

Упражнение

- Устанавливаем node.js
- проверяем версию.
- в консоли должна работать: node -v
- напишите Вашу версию в чат

- Пакетный менеджер NodeJS.
- Не единственный.
- Конфигурационный файл – package.json.

Демо

OTUS

```
npm init
```

Немного о import/export

- import/export – экспериментальная поддержка
- И только в файлах с расширением .mjs (Michael JackSon)
- Изначально использовался (и используется часто) module.export/require

export

```
// lib/math.js
const pi = 3.14;

module.exports = {
    pi,
    exp: (x) => Math.exp(x)
}

//  app.js
const pi = require('lib/math').pi;
const exp = require('lib/math').exp;

console.log("e^{π} = " + exp(pi))
```

Default export

OTUS

```
// lib/exponent.js
const exp = (x) => Math.exp(x)

module.exports = exp;

//  app.js
const exp = require('lib/exponent');

console.log(exp(2));
```

Демо

OTUS

export/import

Ваши вопросы?

05

Express

OTUS

Упражнение:
Установим Express?

```
// package.json
{
  "private": true,
  "dependencies": {
    "express": "4.16.3"
  }
}
```

Или

```
npm init
npm i express -save
```

OTUS

Упражнение:
Установим Express?

```
// server.js
var express = require('express');
var app = express();

app.get('/', function(req, res){
    res.send('Hello World');
});

app.listen(3000);
```

и npm start
и откроем localhost:3000

OTUS

Упражнение:
Установим Express?

// server.js и npm start localhost:3000/hello?name=Ivan

```
var express = require('express');
var app = express();

app.get('/hello', function(req, res){
    res.send('Hello ' + req.query.name);
});

app.listen(3000);
```

OTUS

Несколько слов про методы express

```
app.METHOD(path, callback [, callback ...])
    app.get('/', function (req, res) {
    res.send('GET request to the homepage')
})
```

GET - HEAD Same as GET, but only transfer the status line and header section
POST Perform resource-specific processing on the request payload
PUT Replace all current representations of the target resource with request payload
DELETE Remove all current representations of the target resource
OPTIONS Describe the communication options for the target resource
PATCH A set of changes described in the request entity be applied to the resource

app.all() for all types of requests

HTTP-методы

GET	получение страниц	получение entity
POST	отправка данных формы	создание entity
PUT	---	обновление entity
PATCH	---	изменение entity (RFC)
DELETE	---	удаление entity
HEAD	хитрый	хитрый
OPTIONS	хитрый	хитрый
TRACE	не используется	не используется

OTUS

Пример RESTful стиля API

GET /person/ - все

GET /person/:id – получение конкретного

POST /person - создание

PUT /person/:id – изменение/замена

PATCH /person/:id – изменение/частичное обновление

DELETE /person/:id - удаление

OTUS

Несколько слов про методы express

```
app.METHOD(path, callback [, callback ...])
    app.get('/', function (req, res) {
    res.send('GET request to the homepage')
})
```

GET - HEAD Same as GET, but only transfer the status line and header section
POST Perform resource-specific processing on the request payload
PUT Replace all current representations of the target resource with request payload
DELETE Remove all current representations of the target resource
OPTIONS Describe the communication options for the target resource
PATCH A set of changes described in the request entity be applied to the resource

app.all() for all types of requests

Request и response

- Могли уже заметить объекты request и response
- С помощью них можно получать параметры запроса
- И управлять ответом
- Их можно обогащать

Request

https://expressjs.com/ru/api.html

- req.app
- req.baseUrl
- req.body
- req.cookies
- req.fresh
- req.hostname
- req.ip
- req.ips
- req.method
- req.originalUrl
- req.params
- req.path
- req.protocol
- req.query
- req.route
- req.secure
- req.signedCookies
- req.stale
- req.subdomains
- req.xhr

- req.accepts()
- req.acceptsCharsets()
- req.acceptsEncodings()
- req.acceptsLanguages()
- req.get()
- req.is()
- req.param()
- req.range()

Request.query

- В request.query лежат все параметры запроса

GET http://localhost:3000/hello?name=Ivan

{name: ‘Ivan’}

POST http://localhost:3000/hello
name=Ivan

{name: ‘Ivan’}

Упражнение

```
var express = require('express');
var app = express();

app.get('/hello', (req, res) => {
   res.send('Hello ' + req.query.name);
});

app.listen(3000, () => console.log('Hello from 3000'));
```

```
npm start
```

Откройте http://localhost:3000/hello?name=Ivan

Упражнение

```
var express = require('express');
var app = express();

app.get('/hello/:id', (req, res) => {
   res.send('Hello ' + req.params.id);
});

app.listen(3000, () => console.log('Hello from 3000'));



npm start

Откройте http://localhost:3000/hello/790
```

Response

- Response – специальный объект для управления ответом
- Это действительно мощнейший инструмент, который позволяет сделать много чего
- Задать Cookie, вернуть файл-JSON.
- Обработка всегда должна заканчиваться res.send()

Response

Properties

res.app
res.headersSent
res.locals

Methods

res.append()
res.attachment()
res.cookie()
res.clearCookie()
res.download()
res.end()
res.format()
res.get()
res.json()
res.jsonp()
res.links()
res.location()
res.redirect()
res.render()
res.send()
res.sendFile()
res.sendStatus()
res.set()
res.status()
res.type()
res.vary()

Упражнение

Один метод – отдаёт GET /person {name: ‘Ivan’}

Второй метод – просто статус-код NO-CONTENT

```
res.send({...}) или res.json()

res.sendStatus(...)



https://expressjs.com/ru/api.html
```

Обслуживание статического веб-контента

OTUS

- Статический контент таким способом не отправляют
- Обычно используют middleware

```
app.use(express.static('public'));
app.use(express.static('files'));
```

```
app.use('/static', express.static('public'));
```

```
http://localhost:3000/images/kitten.jpg
http://localhost:3000/css/style.css
http://localhost:3000/js/app.js
http://localhost:3000/images/bg.png
http://localhost:3000/hello.html
```

```
http://localhost:3000/static/images/kitten.jpg
http://localhost:3000/static/css/style.css
http://localhost:3000/static/js/app.js
http://localhost:3000/static/images/bg.png
http://localhost:3000/static/hello.html
```

Роутинг

- Обозначения маршрутов – не просто обозначения
- С помощью них можно получать параметры
- Крайне полезно в RESTful API
- Используется библиотека path-to-to-regexp

- https://www.npmjs.com/package/path-to-regexp
- https://expressjs.com/ru/guide/routing.html

Упражнение

Попробуйте поменять роут и отправить запрос на:

```
app.get('/hel+o/:id', (req, res) => {

app.get('/hell?o/:id', (req, res) => {

app.get('/hello/(:id)?', (req, res) => {

app.get(/.*/', (req, res) => {
```

OTUS

Express Middlewares

Упражнение

Требуется добавить логирование времени запроса в каждый (!) метод.

Как это сделать?

```
app.get('/hello', (req, res) => res.send({name: 'Ivan'}));

// методов много!

app.listen(3000);
```

- Это промежуточный слой для обработки запросов
- Служит именно для того, чтобы не добавлять код в бизнес-логику
- Бизнес-логика остаётся чистой
- Функциональность добавлена прозрачно

Пример middlewares

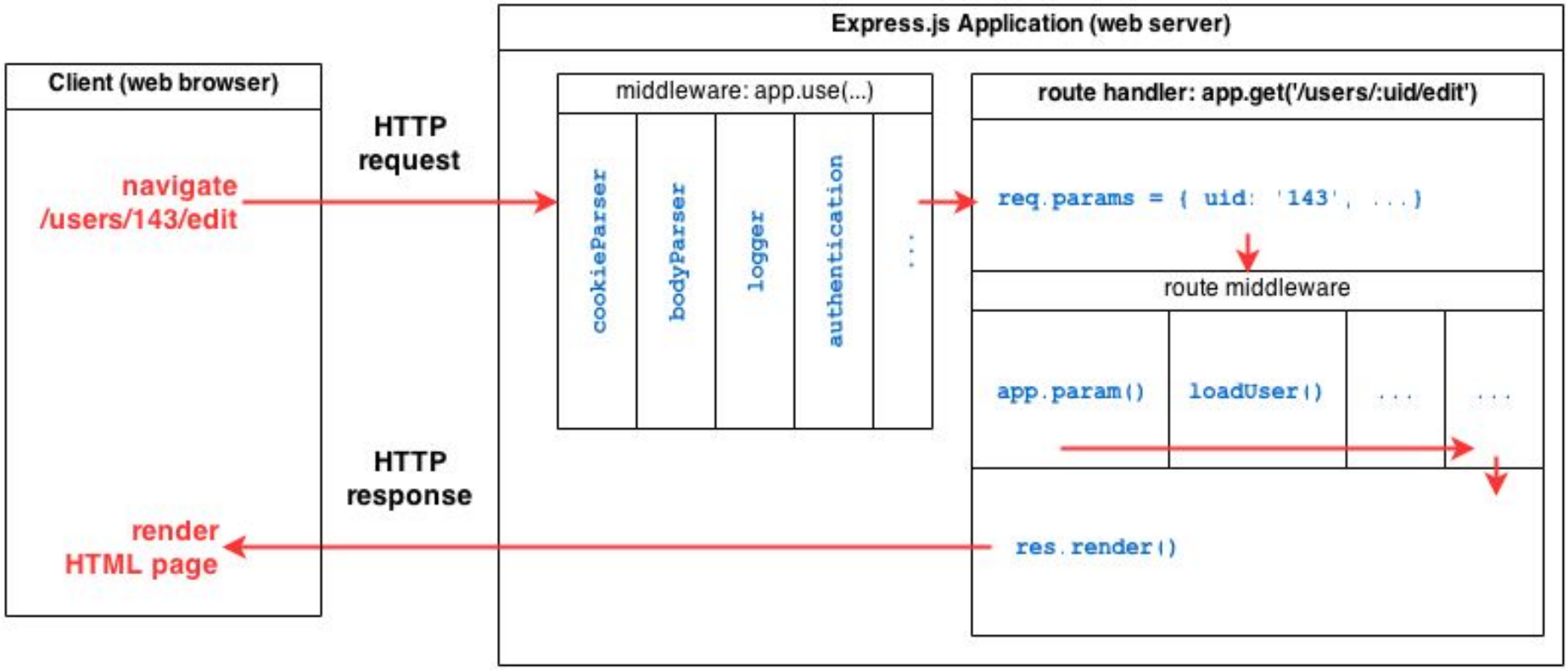
OTUS

```
app.use((req, res, next) => {
    console.log(new Date().toISOString());
    next();
});


app.get('/person', (req, res) => res.send({name: 'Ivan'}));


app.listen(3000);
```

Request и response

- request и response – это всё те же объекты
- Самая фишка в том, что эти объекты можете модифицировать
- next – специальный callback – который знаменует передачу управления
- Кстати, можно совсем не вызывать next(), если middleware управляет вызовом

Middlewares



Body-parser

OTUS

```
const express = require('express'),
      app = express(),
      bodyParser = require('body-parser');

// support parsing of application/json type post data
app.use(bodyParser.json());

//support parsing of application/x-www-form-urlencoded post data
app.use(bodyParser.urlencoded({ extended: true }));
```

Body-parser urlencoded

OTUS

```
app.post('/post-test', (req, res) => {
    console.log('Got body:', req.body);
    res.sendStatus(200);
});
```

```
$ curl -d "username=scott&password=secret&website=stackabuse.com" -X POST http://localhost:8080/post-test
OK
```

```
$ node index.js
Started server at http://localhost:8080!
Got body: { username: 'scott',
  password: 'secret',
  website: 'stackabuse.com' }
```

Body-parser json

OTUS

```
app.post('/post-test', (req, res) => {
    console.log('Got body:', req.body);
    res.sendStatus(200);
});
```

```
POST /post-test HTTP/1.1
Host: localhost:8080
Content-Type: application/json
Content-Length: 69

'{"username":"scott","password":"secret","website":"stackabuse.com"}'
```

```
$ node index.js
Started server at http://localhost:8080!
Got body: { username: 'scott',
  password: 'secret',
  website: 'stackabuse.com' }
```

Static web

OTUS

```
var express = require('express')
var path = require('path')
var serveStatic = require('serve-static')

var app = express()

app.use(serveStatic(path.join(__dirname, 'public-optimized')))
app.use(serveStatic(path.join(__dirname, 'public')))
app.listen(3000)
```

Упражнение

Сделать хранилище пользователей

API для массива, id – индекс в массиве

Порядок роутов важен!

```
GET /person/1
response: {id: 1, name: "Ivan", age: 15}

GET /person/create?name=Irina&age=18
response: {id: 2, name: "Irina", age: 18}

Дополнительно – POST /person (+body-parser)
```

Ваши вопросы?

Домашнее задание

НЕТ

Ваши вопросы?

Пожалуйста, пройдите опрос

https://otus.ru/polls/43435/

Спасибо за внимание!

Не бойтесь JS!