


Node.js

03.03.2022

Marek Nagy

Node.js

- webový server
 - náhrada apache
 - podporuje HTTP protokol aj WebSocket
 - využíva JavaScript (knihnica **V8**)
- **event-driven** architektúra
- **non-blocking** I/O
- real-time distribuované aplikácie
 - pomocou websocketu



Využíva aj Chrome.
Vyvíja Google ako
open source
knihnicu.

Node.js

- vytvorenie súboru **projekt.js**

```
for (let i = 1; i < 10; i++) console.log (i);
```

- spustenie

```
node projekt.js
```

Moduly

- pripájanie „knižníc“ / balíčkov / súborov
- hlavný kód

```
let circle = require ('./circle.js');  
console.log ('Obsah kruhu je :' + circle.area (4) );
```

- modul - súbor **circle.js**

```
const PI = Math.PI;  
module.exports.area = function (r) {return PI*r*r};
```

require

- vráti hodnotu z **module.exports**
 - môže to byť hodnota, funkcia, trieda, objekt
- načíta (spustí) sa iba raz, potom vracia iba odkaz na hodnotu **module.exports** v pamäti

```
const circle1 = require ('./circle.js');  
const circle2 = require ('./circle.js');  
  
console.log (circle1 === circle2); // true
```

Obalenie modulov

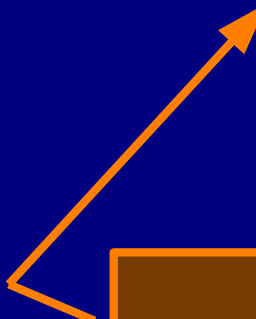
- automaticky (potajme) anonymnou funkciou

```
(function (exports, require, module, __filename, __dirname) {  
    // tu je kód zo súboru - modulu.  
    // Napr.:  
  
    var x=3; let y=4; const a=17; // lokálne premenné modulu  
    r=10; // globálna premenná !  
});
```

- globálne premenné vidieť vo všetkých moduloch
- premenné modulov sú lokálne

Premenné modulu

- __filename** – absolútna cesta súboru modulu
- __dirname** – cesta ku súboru
- module** – hlavný objekt modulu
- exports** – priradená hodnota z **module.exports**
 - pozor, keď sa zmení **module.exports**
- require** – funkcia require



Na začiatku je
`exports=module.exports={}`. Čo
sa môže zmeniť.

Objekt **module**

.exports

- na začiatku prázdny objekt, možno predefinovať
- „návratová“ hodnota modulu

.children

- zoznam požadovaných modulov (objektov **module**)

.require.main

- odkaz na **module**, ktorý prvý požadoval tento modul

Typy modulov

- **základný** (core)
 - hľadá priamo v priečinkoch inštalácie
 - má prednosť
- **súbor**
 - hľadá v rôznych priečinkoch daný súbor
- **priečinkov**
 - hľadá preddefinované názvy súborov v danom priečinku

modul - súbor

- názov začína /
 - absolútna cesta
- názov začína ./ , ../
 - hľadá v aktuálnom priečinku
- inak
 - hľadá v priečinku node_modules
 - alebo vo vyššom node_modules, ... vyššom,
- skúša aj prípony .js, .json, .node
- keď súbor nenájde, hodí chybu

modul - priečinok

- hľadá súbory v danom poradí

package.json - JSON objekt, kde "main" vlastnosť obsahuje meno súboru

index.js - načíta ako JavaScript

index.json - načíta ako JSON objekt

index.node - načíta ako binárku modulu

- keď nenájde, hodí chybu

package.json


- info o module – priečinku

```
{  'name': 'Nazov projektu',  
  'main': './moj.js'  
}
```


npm

- balíčkovací manager pre JavaScript
- uvažujú sa moduly ako priečinky
- balíčky musia mať správne nastavený súbor `package.json`
- pomocník na vytvorenie `package.json`:

```
mkdir mojModul  
cd mojModul  
npm init
```



Podobne aj v inom operačnom systéme. Vytvoriť a vôjsť do priečinka.

npm install

- nainštalovanie balíčka zo siete
- vytvorí sa podpriechinok **node_modules** v aktuálnom priečinku

```
npm install externy_modul
```

Ak sa neuvedie žiaden balíček, pozrie sa na uložené závislosti v package.json a tie doinštaluje.

- prepínač **--save-prod** uloží sa závislosť do package.json

```
npm install externy_modul --save-prod
```

Defaultne je tento prepínač zapnutý.

- prepínač **--no-save** neuloží závislosť do package.json

```
npm install externy_modul --no-save
```

npm skripty

- **package.json** obsahuje záznam **scripts**, kde sú uvedené shell príkazy napr. na spustenie...

```
"scripts": {  
  "start": "node server.js &",  
  "stop": "killall node"  
},
```

Samozrejme, že
v tom druhom OS
toto treba prerobiť,
alebo nechať
prázdne.

```
"scripts": {  
  "start": "test ! -f /tmp/server.pid && { node server.js &>>  
/tmp/server.log & echo $! > /tmp/server.pid; }; echo 'Started'",  
  "stop": "test -f /tmp/server.pid && { kill $(cat  
/tmp/server.pid); rm /tmp/server.pid; }; echo 'Stopped'"  
},
```

- následne stačí volať

```
npm start  
npm stop
```


npm uninstall

- odinštalovanie modulu

```
npm uninstall externy_modul
```

npm update

- zisťovanie aktuálnosti modulov

```
npm outdated
```

- update-ovanie modulov

```
npm update
```


modul Console

- globálna premenná **console** je už inicializovaná hodnotou **require** ('console')

```
//console = require ('console'); // už netreba zadávať

console.log ('Výpis na štandardný výstup');
console.error ('Výpis do chybového výstupu');

// Zmera čas výpočtu
console.time ('znacka');
....
console.timeEnd ('znacka');
```

Podmienené chybové výpisy

- index.js

```
const util = require('util');  
const log = util.debuglog ('znacka2');  
log ('Chybový výpis');
```

Vyrobí funkciu,
ktorá sa bude
blokovať podľa
danej značky.

- nastaviť premennú prostredia NODE_DEBUG

```
$ export NODE_DEBUG=znacka1,znacka2,znacka3  
$ node index.js  
ZNACKA2 5802: Chybový výpis
```

Vypísanie objektu

- konverzia objektu na textový výpis do konzoly
- index.js

```
const util = require('util');  
  
let obj = {a:3, c:function(x){return 3}};  
  
let txt = util.inspect (obj, {colors:true});  
  
console.log (txt);
```

- farebný výstup

```
$ node index.js  
{ a: 3, c: [Function: c] }
```


modul **File System**

- funkcie na prácu so súbormi a priečkami
 - obsah ako **utf8** reťazec alebo **Buffer** pole
- vykonanie funkcie:
 - **synchronne** (Sync verzie funkcií)
 - blokovanie aplikácie
 - try, catch chytanie chýb
 - **asynchrónne**
 - po vykonaní sa volá callback funkcia
 - súslednosť nadväzujúcich príkazov musí byť povernáraná do callbackov

Niektóre metody FS

`readFile` (file [,options], **callback**)

`readFileSync` (file [,options])

`writeFile` (file, data [,options], **callback**)

`writeFileSync` (file, data [,options])

`readdir` (path [, options], **callback**)

`readdirSync` (path [, options])


Načítanie súboru

- synchrónna verzia

```
const fs = require('fs');

try {
  let txt = fs.readFileSync ('data.txt', {encoding: 'utf8'});
  console.log (txt);
}

catch (err) {
  console.log (err.message);
}
```

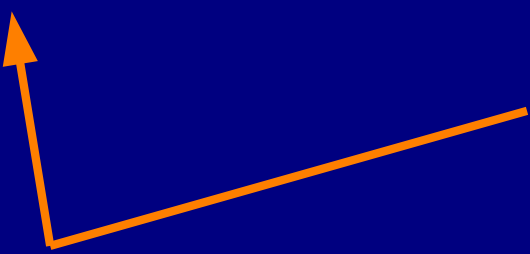


Ak sa neuvedie
kódovanie, výsledkom
budú binárne dáta
(inštancia triedy Buffer).

Načítanie súboru

- asynchrónna verzia


```
const fs = require('fs');  
  
fs.readFile ('data.txt', {encoding:'utf8'}, (err, data) => {  
  
    if (err)  
        console.log (err.message);  
  
    else  
        console.log (data);  
  
});
```



Neskôr sa naučíme, že
existuje elegantnejší
mechanizmus async funkcií.

Callback hell

```
fs.readFile ('data1.txt', {encoding: 'utf8'}, (err, data) => {  
  if (!err) {  
    fs.readFile ('data2.txt', (err, data) => {  
      if (!err) {  
        fs.readFile ('data3.txt', (err, data) => {  
          if (!err) {  
            fs.readFile ('data4.txt', (err, data) => {  
              if (!err) {  
                console.log ('Hura');  
              }  
            });  
          }  
        });  
      }  
    });  
  }  
});
```



modul HTTP

- realizuje:
 - dotazy na server
 - samostatný web server
- komunikuje protokolom HTTP

```
const http = require ('http');

const server = http.createServer ((req, res) => {
  // Požiadavky v req
  // Výstup sa píše do res

  res.writeHead (500);
  res.end ('Error');
});

server.listen (9000);
```

Jednoduchý HTTP server

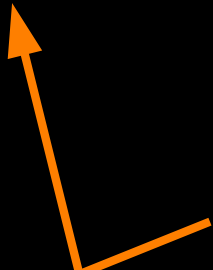
```
const http = require ('http');
const fs = require ('fs');

const server = http.createServer( (req, res) => {

  // Odpovedá vždy obsahom súbora

  fs.readFile ('index.html', (err, data) => {
    if (err) {
      res.writeHead (500);
      res.end ('Error loading index.html');
    }
    else {
      res.writeHead (200, {'Content-Type': 'text/html'});
      res.end (data);
    }
  });
});

server.listen (9000, () => {
  console.log ('Listen on port 9000');
});
```



Na prevádzkovanie HTTP servera je ideálnejší modul **express**, o ktorom bude samostatná prednáška.

Vzniká alternatíva: Deno

- objavila sa v roku 2020
- tiež využíva knižnicu **V8**
 - pre **JavaScript** a **WebAssembly**
 - Chrome, node.js, deno, ...
- má aj integrovaný kompilátor pre **TypeScript**
 - kompiluje na JavaScript kód
- nepoužíva npm
 - balíky priamo cez URL (syntax pre **import**)
- počiatočné limitácie
 - napr. socket.io ešte nie celkom podporovaný, a čo express?, ...
 - nie je kompatibilita s API pre node.js

Ďakujem za pozornosť