SZERVER OLDALI JAVASCRIPT

11. hét

core modulok

MIÉRT MOST?

Elegendő tapasztalat és tudás gyűlt össze bennetek, hogy ne csak egy "api listát" lássunk, hanem ténylegesen meg is maradjon valami. :D

Pár hasznos funkciót mutatok, amit nem mindenki fog használni.

MIRŐL LESZ SZÓ

- Cluster module
- Events
- FS
- Stream
- HTTP, HTTPS
- OS

CLUSTER

Node egy threadon fut - mit tegyünk ha skáláznánk több magra: **cluster**

Saját magát tudja forkolni egyszerűen

```
var cluster = require('cluster');
var http = require('http');
var numCPUs = require('os').cpus().length;

if (cluster.isMaster) {
  // Fork workers.
  for (var i = 0; i < numCPUs; i++) {
    var worker = cluster.fork();
  }
} else {
  // Workers can share any TCP connection
  http.createServer(function(req, res) {
    res.writeHead(200);
    res.end("hello world\n");
  }).listen(8000);
}
```

CLUSTER

Hogyan kommunikáljak threadek között?

Master oldalán:

```
worker.on('message', function(msg) {
    worker.send('Hey worker');
});
```

Worker oldalán:

```
process.on('message', function(msg) {
    console.log('From master', msg);
});

process.send('To master: Hey');
```

EVENTS

A node egyik működési köve: eventek

Feliratkozás:

```
var EventEmitter = require('events');

var server = new EventEmitter();

server.on('connection', function (stream) {
  console.log('someone connected!');
});
```

Esemény meghívása:

```
server.emit('connection',theStream);
```

EVENTS

Egyszeri meghívásra is fel lehet iratkozni:

```
server.once('connection', function (stream) {
  console.log('only once thanks');
});
```

Le is lehet iratkozni:

```
var callback = function(stream) {
  console.log('someone connected!');
};
server.on('connection', callback);
// ...
server.removeListener('connection', callback);
```

EVENTS - ADMINISZTRÁCIÓ

Listenerek lekérdezése:

```
console.log(server.listeners('connection'));
// [ [Function] ]
```

Összes feliratkozás eltávolítása

```
server.removeAllListeners('connection');
```

Feliratkozás limitálása

```
server.setMaxListeners(10);
```

EVENTS

Ez is **szinkron**, így lényegében csak "szépítés"!!!

Sosem hívódik meg a console.log

```
var EventEmitter = require('events');

setTimeout(function(){console.log('im here');},10);

var emitter = new EventEmitter();
emitter.on('msg',function(something){ });

while (true){
    emitter.emit('msg','boo');
}
```

EVENTS

Ha meg akarsz törni egy event loopot, akkor a setImmediate, nextTick amit hívnod kell.

```
var EventEmitter = require('events');

setTimeout(function(){console.log('im here');},10);

var emitter = new EventEmitter();
emitter.on('msg',function(something){ });

function loop(){
    emitter.emit('msg','boo');
    setImmediate(loop);
}
loop();
```

FS

Minden ami fájlrendszer. Nézzük meg miket tud:

KÖNYTÁRAK SZINTJÉN

- **könyvtár létrehozás:** fs.mkdir(path[, mode], callback)
  - mode unix alapokon izgalmas: 0777
- **könyvtár own:** fs.chown(path, uid, gid, callback)
- **könyvtár átnevezés:** fs.rmdir(path, callback)
- **könyvtár törlés:** fs.unlink(path, callback)
- **könyvtár tartalmának betöltése:** fs.readdir(path, callback)
  - callback: (err, files) a . és .. nélkül

FS

FÁJL SZINTJÉN

- **fájl megnyitás:** fs.open(path, flags[, mode], callback)
  - C++ alapok alapján lényegében:
    - flags: r, w, a stb.
    - mode: fájl mód - 0666
- **fájl olvasás:** fs.read(fd, buffer, offset, length, position, callback)
  - position: null ha az aktuális cursortól
  - callback: (err, bytesRead, buffer)

FS

FÁJL SZINTJÉN

- **fájl írás - bináris adatok:** fs.write(fd, buffer, offset, length[, position], callback)
  - callback: (err, written, buffer)
- **fájl írás - szöveges mód:** fs.write(fd, data[, position[, encoding]], callback)
- **fájl bezárása:** fs.close(fd, callback)
- **fájl végére írás egy lépésben:** fs.appendFile(file, data[, options], callback)
  - options: encoding, mode, flag vagy kombinációi

FS

A végére valami izgalmas is maradt:

WATCH

Fel lehet iratkozni fájlrendszer változásaira:

- **fájlok/könyvtárak figyelése:** fs.watch(filename[, options][, listener])
  - options: { persistent: true, recursive: false }
    - persistent: a végtelenségig figyel (node mikor áll le)
    - recursive: ...
  - listener: (event, filename) pl: console.log(event,filename) kimenetek:

```
- change code/watch.js
- rename code/watch2.js        (ez a mentés pl)
```

FS

WATCH

- **fájlok figyelése:** fs.watchFile(filename[, options], listener)
  - callback: (currStat, prevStat) fájl statisztikák (létrehozás, módosítás, stb)
- **fájl figyelés felfüggesztése:** fs.unwatchFile(filename[, listener])

STREAM

Olvasásra / írásra képes abstract interface.

4 típus van:

- Readable
- Writable
- Duplex: az előző 2
- Transform

Fájlok, udp / tcp kapcsolatok, bármi absztakt.

STREAM

A streameke esetében fel kell iratkozni az egyes eseményekre (eventek).

READABLE STREAM EVENTEK

```
var readable = getReadableStreamSomehow();
readable.on('data', function(chunk) {
  console.log('got %d bytes of data', chunk.length);
});
readable.on('end', function() {
  console.log('there will be no more data.');
});
```

Lehetséges az adatok küldését leállítani / újraindítani:

```
readable.pause();
readable.resume();
```

STREAM

WRITEABLE STREAM

Írni lehet...

```
writable.write(chunk[, encoding][, callback])
writable.end([chunk][, encoding][, callback])
```

Itt is vannak eventek:

```
writable.on('drain', function() {
  console.log('erdemes ujra irni');
});
writable.on('finish', function() {
  console.log('minden bit odaert');
});
```

STREAM

Viszont ami még izgalmasabb, ezeket egymásba lehet **drótozni**.

Egy TCP-ből olvasást egy fájlba lehet irányítani, vagy éppen egy fájl olvasást egy gzip Transformba ÉS egy fájlba, az gzip transformot pedig egy másik fájlba, stb.

```
var r = fs.createReadStream('file.txt');
var z = zlib.createGzip();
var w = fs.createWriteStream('filecopy.txt');
var wzip = fs.createWriteStream('file.txt.gz');
var zout = r.pipe(z);
zout.pipe(wzip);
r.pipe(w);
r.on('end',function(){
    console.log('olvasas vege');
});
```

HTTP ÉS HTTPS

Listen része már ismerős:

```
var http = require("http");
var server = http.createServer(function(request, response) {
    //....
});
```

HTTP ÉS HTTPS

De hívást kezdeményezni is lehet!

```
var req = http.request({hostname: 'www.google.com',
port: 80, path: '/', method: 'GET'}, function(res) {
  console.log('STATUS', res.statusCode, 'HEADERS', res.headers);
  res.setEncoding('utf8');
  var chunk = '';
  res.on('data', function (part) {
      chunk+=part;
  });
  res.on('end', function() {
    console.log('BODY: ' + chunk);
  })
});
req.on('error', function(e) {
  console.log('problem with request: ' + e.message);
});
req.end();
```

HTTP ÉS HTTPS

POST is működik, csak nevezebb összehozni :)

```
var postData = querystring.stringify({
  'msg' : 'Hello World!'
});
var options = {
  hostname: 'www.google.com',
  port: 80,
  path: '/upload',
  method: 'POST',
  headers: {
    'Content-Type': 'application/x-www-form-urlencoded',
    'Content-Length': postData.length
  }
};
var req = http.request(options, function(res) { /* ... */ });
req.write(postData);
req.end();
```

OS

OS szintű, inkább izgalmas mint hasznos funkciók:

- **könyvtárak:** os.tmpdir(), os.homedir()
- **egyszerű statisztikák:** os.uptime(), os.loadavg(), os.totalmem(), os.freemem(), os.cpus(), os.networkInterfaces()
- **hasznos tulajdonságok**: os.hostname(), os.type(), os.platform(), os.arch(), os.release()
- os.endianness(), os.EOL

ERROR.STACK TRÜKK

Ha szeretnéd megtudni, hogy egy adott pontra "hogyan jutott" az irányítás:

```
console.log ((new Error()).stack)
```

Hasznos tud lenni, ha egy hibával / izgalmas működéssel találkozunk, és tudni szeretnénk milyen callstack van mögötte

MIT FOGUNK GYAKORLATON CSINÁLNI?

Megnézzük ezeket példa szinten is.