SZERVER OLDALI JAVASCRIPT

7. hét

Promise, async, error first callback, grunt

ELŐADÁS TÉMÁJA

Pár izgalmasabb javascript / node specifikus dolog.

A nyelvi elemek előadásra épít erősen...

FUNCTIONÖK / CALLBACKEK

```
module.exports = function (objectrepository) {
  return function (req, res, next) {
    userModel.findOne({}, function (err,results){
        res.tpl.users = results;
        return next();
    });
    return next();
  };
};
```

A **function** nagyon gyakran használt nyelvi elem.

EGYSZERŰ PÉLDA - FÁJL BEOLVASÁS

```
fs.readFile('config.json',
        function (error, text) {
            if (error) {
                console.error('Error while reading config file');
            } else {
                try {
                    var obj = JSON.parse(text);
                    console.log(JSON.stringify(obj, null, 4));
                } catch (e) {
                    console.error('Invalid JSON in file');
                }
            }
        });
```

"Sima" aszinkron és szinkron funkciók egymás után.

EGYSZERŰ PÉLDA - FÁJL BEOLVASÁS

Használjunk promiset!

```
readFilePromisified('config.json')
.then(JSON.parse)
.then(function (obj) {
    console.log(JSON.stringify(obj, null, 4));
})
.catch(function (reason) {
    console.error('An error occurred', reason);
});
```

PROMISE

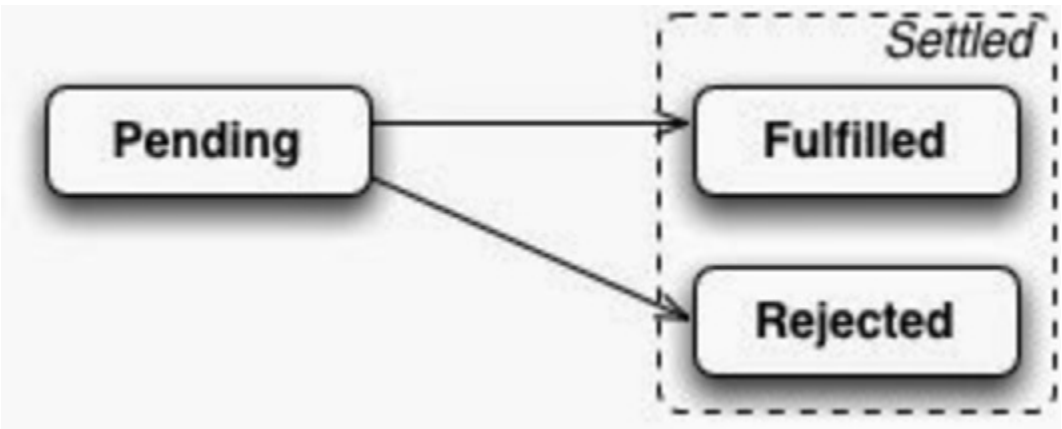
Egy oszályba burkolt függvény, ami az állapotváltását láncolható módon tájékoztatja a rá feliratkozókat. ES6-ban natív nyelvi elem!!

```
    var promise = new Promise(
        function (resolve, reject) {
            ...
            if (...) {
                resolve(value);
            } else {
                reject(reason);
            }
        });

promise
.then(
    function(a){console.log('Resolve:' + a);},
    function(b){console.log('Reject:' + b);})
.catch(function(e){console.error(e);});
```

PROMISE

Siker esetén resolve, hiba esetén reject. Ha egyiket sem hívod, akkor végtelenségig vár!



és dobott kivételeket is lehet kezelni (ha nincs reject ág then-ben, akkor a catchig jut el a hibakezelés)

PROMISE

Láncolhatóak ha egy függvény promiset ad vissza, vagy ha szinkron.

Ha bármelyik rejectet hív, megtörik a lánc.

```
function promiseFv(item){
    return new Promise(function (resolve, reject) {
        resolve(item+'p');
    });}
```

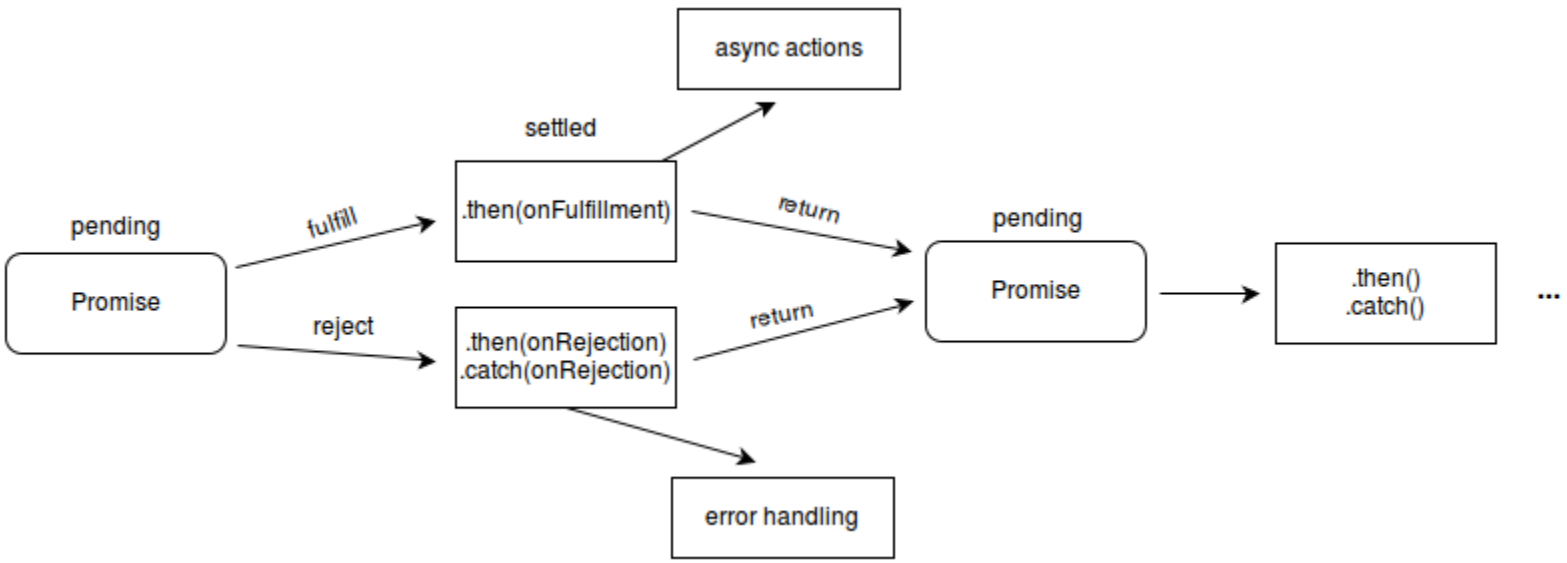
PROMISE LÁNC

```
function promiseFv(item) {
  return new Promise(function (resolve, reject) {
    resolve(item + 'p');
  });
}
function normalFv(item) { return item + 'n'; }

promiseFv('1')
  .then(normalFv)
  .then(promiseFv)
  .then(function (item) {return item + '2';})
  .then(console.log)
  .catch(function (e) {
      console.log('error:' + e);
});
```

Ezt írja ki: **1pnp2**

PROMISE LÁNC



Ha a láncban bárhol hiba dobódik, akkor az az első catch-ig fog elérni.

Elég egy catch, kevesebb hibakezelési ág, ez jó!

PROMISE.ALL

Több promise egy várakozásban összefogva - vagy megvárja mindegyik sikerét, vagy az első kudarcot.

```
var p1 = Promise.resolve(3);
var p2 = 1337;
var p3 = new Promise(function(resolve, reject) {
  setTimeout(resolve, 100, "foo");
});

Promise.all([p1, p2, p3]).then(function(values) {
  console.log(values); // [3, 1337, "foo"]
});
```

PROMISE.RACE

Több promise, amelyik gyorsabban befejeződik, az adódik át a thenben (a többi is fut / lefut, nem szakítódik meg).

```
var p1 = new Promise(function(resolve, reject) {
    setTimeout(resolve, 500, "one");
});
var p2 = new Promise(function(resolve, reject) {
    setTimeout(resolve, 100, "two");
});

Promise.race([p1, p2]).then(function(value) {
  console.log(value); // "two"
  // Both resolve, but p2 is faster
});
```

PROMISE.RESOLVE, PROMISE.REJECT

Szinkron function / érték becsomagolása promiseba, van ennek értelme. :)

```
function foo(){
    return Promise.resolve('alma');
}
function bar(){
    return Promise.reject('alma');
}

foo.then(bar).catch(console.log);
```

PROMISE

Bizonyos esetekben ez kell, bizonyos esetekben normál callbackek, illetve beszéljünk az **async npm modulról**.

```
npm i async
```

ASYNC

Async.js

Listákon végrehajtott tömeges műveletek, control flow műveletek, általános async műveletek - szinte mindenre van valamilyen megérhető recept.

ASYNC - EACH, EACHSERIES, EACHLIMIT

- each(arr, iterator, [callback])
- eachSeries(arr, iterator, [callback])
- eachLimit(arr, limit, iterator, [callback])

```
var lista = [1,2,3];
async.each(lista, function(item, callback) {
  console.log('Processing:'+item);
  setTimeout(function(){ console.log('Processing done:'+item);
  callback(null);  },Math.random()*1000);
}, function (err){ console.log('All done'); });
```

Eredmény:

```
Processing:1
Processing:2
Processing:3
Processing done:3
Processing done:1
Processing done:2
All done
```

ASYNC - MAP (MAPSERIES, MAPLIMIT)

map(arr, iterator, [callback])

```
var lista = [1,2,3];
async.map(lista, function (item, callback) {
  callback(null, item * item);
}, function (err, results) {
  console.log(results);
});
```

Eredmény:

```
[ 1, 4, 9 ]
```

ASYNC - SERIES

series(tasks, [callback])

```
async.series([
    function(callback){
        callback(null, 'one');
    },
    function(callback){
        callback(null, 'two');
    }
],
function(err, results){
    console.log(results);
});
```

Eredmény:

```
['one', 'two']
```

ASYNC - PARALLEL

parallel(tasks, [callback])

```
async.parallel([
    function(callback){
        setTimeout(function(){
            callback(null, 'one');
        }, 200);
    },
    function(callback){
        setTimeout(function(){
            callback(null, 'two');
        }, 100);
    }
],
function(err, results){
    console.log(results);
});
```

Eredmény:

```
['one', 'two']
```

ASYNC - WATERFALL

waterfall(tasks, [callback])

```
async.waterfall([
    function(callback) {
        callback(null, 'one', 'two');
    },
    function(arg1, arg2, callback) {
        //arg1: one, arg2: two
        callback(null, 'three');
    },
    function(arg1,callback) {
        //arg1: three
        callback(null, 'done','done2');
       }
  ],
  function(err, result, result2){
      console.log(result,result2);
  }
```

Eredmény:

```
done done2
```

ASYNC

És még sokan mások...

Tessék dokumentációt olvasni hozzá:
https://github.com/caolan/async

Instrumentáláshoz az egyik legjobb library.

MINDEN FELADATFÜGGŐ:

- async
- promise (promisify)
- callback
- sima függvények
- aszinkron fv

ERROR FIRST CALLBACK

Sok helyen láttuk már a mintázatot:

```
function callback(error,result){
    ...
}
```

A legtöbb esetben a node ezt használja hibajelzésre async esetben.

ERROR FIRST CALLBACK

Példa:

```
fs.readFile('/etc/passwd', function (err, data) {
  if (err) throw err;
  console.log(data);
});
```

Ez egy jó mintázat! Ha async akarunk hibát jelezni, tegyük így! A **throw** esetében kaotikusabb megmondani, hol köt ki a hibánk.

ERROR FIRST CALLBACK

Könnyű mockolni/tesztelni is.

Mindig nézzük meg, hogy az általunk használt modul / package / library / whatever mit használ.

Sok a promise és az error first callbackes megoldás is.

GULP

The streaming build system

GULP

Node-ban alapvetően nincs build folyamat, az emberek pedig szeretnek buildelni. :)

Deploy előtti automatikus scriptekről beszélünk ilyenkor általában, de nagyon sok felhasználási mód van még.

GULP

Telepítsük globálisan

```
npm i gulp-cli -g
```

Illetve egy dev package is kell

```
npm i --save-dev gulp
```

Egy speciális fájlt (**gulpfile.js**) használ a folyamatok leírására. A folyamatok külön npm modulokban lehetnek.

```
var gulp = require('gulp');
var csso = require('gulp-csso');

gulp.task('default', function () {
    return gulp.src('./main.css')
        .pipe(csso())
        .pipe(gulp.dest('./out'));
});
```

GULP

Lefuttatva megtörténik a varázslat:

```
$ gulp
[19:30:24] Using gulpfile gulpfile.js
[19:30:24] Starting 'default'...
[19:30:24] Finished 'default' after 45 ms
```

Mindegyik modul más, máshogy kell konfigurálni, paraméterezni, stb.

(Éppen ezért léteznek alternatívák, pl Grunt)

GULP-UGLIFY

Összefűzi és minifyolja a csseket

Telepítés:

```
npm install --save-dev gulp-uglify
```

Modul használata:

```
var uglify = require('gulp-uglify');

gulp.task('compress', function() {
  return gulp.src('lib/*.js')
    .pipe(uglify())
    .pipe(gulp.dest('dist'));
});
```

Paraméterezés:

https://github.com/terinjokes/gulp-uglify

GULP-CSSO

Összefűzi és minifyolja a csseket

Telepítés:

```
npm install gulp-csso --save-dev
```

Modul használata:

```
var gulp = require('gulp');
var csso = require('gulp-csso');

gulp.task('default', function () {
    return gulp.src('./main.css')
        .pipe(csso())
        .pipe(gulp.dest('./out'));
});
```

Paraméterezés:

https://github.com/ben-eb/gulp-csso

GULP-CONCAT

Fájlok összefűzése.

Paraméterezés:

https://github.com/contra/gulp-concat

Példa:

```
var concat = require('gulp-concat');

gulp.task('scripts', function() {
  return gulp.src('./lib/*.js')
    .pipe(concat('all.js'))
    .pipe(gulp.dest('./dist/'));
});
```

GULP

Mindig a feladathoz keresünk eszközt és nem az eszközhöz feladatot!!!

Ez gulp esetében is igaz. Csak akkor használjuk, amikor tényleg szükség van rá.