Application development in JavaScript

Ing. Roman Bronis, PhD.

Faculty of Informatics and Information Technologies

Oct. 2024

Table of Contents

- Intervals
- Arrays
- Objects
- Strings and RegExes
- JSON
- node.js & npm
- JavaScript everywhere
- Modules
- Callbacks
- Promises
- Async/await

Intervals

```
var ival = setInterval(function()..., timeInMs);
clearInterval(ival);
var itim = setTimeout(function()..., timeInMs);
clearTimeout(itim);
```

Arrays I.

```
var a = [1,2,3];
var a = Array(1,2,3);
var a = Array(3);

a.forEach(function(e){console.log(e);});
var l = a.length;
for(var i=0;i<l;i++) {console.log(a[i]);}

var a = [1,2,3,4,5];
a[6] = 7;
console.log(Object.keys(a));
console.log(a.length);
console.log(a);
a.length = 15;
console.log(a);
```

Arrays II.

```
var a = [1,2,3,4,5];
var b = a;
a.pop();
console.log(b);

var b = a.slice();
```

Arrays III.

- array modifiyng operations
  - .push(element/s), .pop(), .unshift(element), .shift()
  - .splice(position, count)
  - .reverse(), .sort((e1,e2) => e1-e2);
- operations returning new array
  - .concat(array2...)
  - .filter(funciton(element){return element>7});
  - ['H','e','l','l','o'].join('');
  - [1,2,3,4,5].map(function(element){return element+1});
  - .slice([start[, end]]);

[0] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Objects I.

```
var a = {};
var a = new Object();

var a = new Object({"key": "value"});
var a = Object.create(protoObject);
var a = {"key": "value"};
```

Objects II.

```
var User = function({name, password, email, score, level})
    this.name = name ? name : null;
    this.password = password ? password : null;
    this.email = email ? email : null;
    this.level = level ? level : null;
    this.score = score ? score : null;
};

var us = new User({name: "Jose", password: "12345"});
User.prototype.hashName = function() {
    const crypto = require("crypto");
    return crypto.createHash('md5').update(this.name)
                                        .digest('hex');
};
for (var key in us) { console.log(key); }
Object.keys(us);
```

Strings and RegExes

```
var wrong = "I'm the badger. ... I'm the one who knocks."
var fixed = wrong.replace('badger','danger');
var intense = fixed.replaceAll("I'm", "I am");
var intense = fixed.replace(/I'm/g,'I am');
```

```
var bool = RegEx.test(str);
var arr = RegEx.exec(str); // capture groups
var arr = str.split(delimiter);
var index = str.search(RegEx);
var arr = str.match(RegEx);
var arr = str.matchAll(RegEx); // capture groups
```

[0] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions

Figure: Douglas Crockford

JSON

- JavaScript Object Notation
- 2001, standardized in 2013
- *JSON* $\subset$ *JavaScriptobjects*
- boolean, number, string, [ ], {}, null
- MIME type application/json

[1] `https://www.youtube.com/watch?v=-C-JoyNuQJs`

HTTP requests

```
jQuery $.ajax() ($.get(), $.post())
fetch()
XMLHttpRequest
```

node.js

- an asynchronous event-driven JavaScript runtime
- `https://nodejs.org/en/download/`
- node file.js
- modules / packages

[1] `https://nodejs.org/en/about/`

Simple HTTP server

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

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Hello World');
});

server.listen(port, hostname, () => {
    console.log('Server running at http://'+hostname+':'+po
});
```

[1] https://nodejs.org/en/about/

npm

- npm init
  - package.json
- npm search <string>
- npm install <package_name>
  - npm install express
  - npm install -g <package_name>

Node module

```
//a. library.js
//b. main.js
0a. let i = 0;
0a. const f = function()  console.log('i: '+(++i));

1a. module.exports = f;
1b. const f = require('./library.js');

2a. exports.f = f;
2b. const f = require('./library.js').f;

3a. export default f;
3b. import f from './library.js';
3c. // change to .mjs or change type to module in package.j
```

Callback

```
var processResult = function(result) {
    console.log(result);
};

var bigCompute = function() {
    var r = 4;
    // this is something taking very long
    setTimeout(function(){
        console.log('result processed')
        processResult(r);
    }, 5000);
    console.log('bigCompute finished');
}
bigCompute();
```

Callback hell

```
doSomething(function(result)
  doSomethingElse(result, function(newResult)
    doThirdThing(newResult, function(finalResult)
      console.log('Got the final result: ' + finalResult);
    , failureCallback);
  , failureCallback);
, failureCallback);
```

1

[1] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises

Promises

- new Promise((resolve,reject)=¿{});
- pending, resolved, rejected
- .then(), .catch()

Promise

```
var bigCompute = new Promise((resolve,reject)=>{
    var r = 4;
    // this is something taking very long
    setTimeout(function(){
        console.log('result processed')
        if(r===4) resolve(r);
        else reject('boom');
    }, 5000);
    console.log('bigCompute finished');
});

bigCompute.then((result)=>{
    console.log(result);
}).catch((err)=>{
    console.error(err);
});
```

Promise

```
var nBigCompute = function(p) {
    return new Promise((resolve,reject) => {
        setTimeout(function(){
            console.log('result processed');
            if(p!==4) resolve(p);
            else reject('boooom');
        }, 5000);
    });
};

nBigCompute(3).then((r)=>{
    console.log(r);
}).catch((e)=>{
    console.error(e);
});
```

Promise chaining

```
doSomething()
.then((result)=>{
    return doSomethingElse(result);
})
.then((newResult) => {
    return doThirdThing(newResult);
})
.then((finalResult) => {
    console.log('Got the final result: ' + finalResult);
})
.catch(failureCallback);
```

2

---

[2] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises

async and await

- async function ...
- await someFunctionResult()
- awain only in async function

async and await

```
async function doTheComputeS() {
    var q = await nBigCompute(1);
    console.log(r);
    var r = await nBigCompute(r+1);
    console.log(q);
}
```

"paralelism"

```
async function doTheComputeP() {
    var [q,r] = await Promise.all([
        nBigCompute(1),
        nBigCompute(2)
    ]);
    console.log(q,r);
}
```