Übungen zu NodeJS

Markus Reichl, 9. Dezember 2017

Inhaltsverzeichnis

1 DNS Lookup

Es soll ein Programm geschrieben werden, das die IP-Adresse(n) einer Domain ermittelt und ausgiebt. Die Domains (beliebig viele) sollen dabei als Kommandozeilenargument angegeben werden.

```
const dns = require('dns')

// First argument is node, second is the script name
// The other arguments shall be looked up
for (let i = 2; i < process.argv.length; i++) {
    let url = process.argv[i]       // Take arguments as urls
    dns.lookup(url, function (err, add) {
        if (err) console.log(err)   // Log errors
        else console.log(url + " :" + JSON.stringify(add))
    })
}
```

Als Beispiele wurden hier die Adressen von `tilehub.io` und `re1.at` abgerufen.

```
% node dnslookup.js tilehub.io re1.at
# Addresses of tilehub.io :"192.30.252.154"
# Addresses of re1.at :"104.28.26.196"
```

2 Auslesen von Dateien

Es sollen zwei Programme geschrieben werden, die alle Dateien in einem gegebenen Verzeichnis synchron bzw. asynchron auslesen und ausgeben. Im ersten Fall soll mit dem Einlesen einer Datei erst begonnen werden, wenn die Ausgabe der vorigen Datei fertig ist, im zweiten Fall soll dies parallel geschehen. Welches der beiden Programme ist performanter?

2.1 Synchron

```
fs = require('fs')

console.time("read-sync")
let files = fs.readdirSync('../', 'utf8')
console.log(files)
console.timeEnd("read-sync")
```

2.2 Asynchron

```
fs = require('fs')

console.time("read-async")
files = fs.readdir('../', 'utf8', function (err, file) {
    // Log error or file
    err ? console.log(err) : console.log(file)
})
console.timeEnd("read-async")
```

2.3 Vergleich

```
% node read-sync.js
# read-sync: 2.429ms

% node read-async.js
# read-async: 0.518ms

% ls
# .idea 0-lesson 1-dnslookup 2-read 3-webserver
# README.md index.js index.txt package.json protokoll.pdf protokoll.tex
```

Wie man sieht erweist sich die asynchrone Implementierung als performanter, was auf die gleichzeitige Abarbeitung von Lesevorgängen zurückzuführen ist. Da bei einer SSD die Lesezugriffe wesentlich schneller abgearbeitet werden ist hier der Unterschied marginal.

3 Einfacher Webserver

Es soll ohne Express oder dergleichen ein Webserver in node.js entwickelt werden, der die angefragte URL als Datei am Server an den Browser zurückgibt.

```
const http = require('http')
const url = require("url")
const fs = require("fs")
// Create the server and listen to 8009
http.createServer( (req, res) => {
    let path = url.parse(req.url).pathname
    // Read file and send response
    fs.readFile("." + path, "UTF-8", (err, str) => {
        if (err) {  // On error
            res.writeHead(404, {'Content-Type': 'text/html'})
            res.write("File not found")
        } else {    // On success
            res.writeHead(200, {'Content-Type': 'text/html'})
            res.write(str)
        }
        res.end()   // End write stream
    })
}).listen(8009)
// Log success
console.log('Server running on http://localhost:8009')
```

Der Webserver ist nach dem Start unter `http://localhost:8009` erreichbar und reagiert auf nicht vorhandene Dateien mit einem 404 Header und der Meldung `File not found`.

```
% node webserver.js
# Server running on http://localhost:8009

% curl localhost:8009/package.json
# {
#   "name": "3-webserver",
#   "version": "1.0.0",
#   "description": "Entwickle einen Webserver in node.js,
#                   der die zur im request angefragten url
#                   gehörende Datei sucht und an den Browser
#                   zurückgibt ohne die Verwendung von express oder dergleichen.",
#   "main": "dnslookup.js",
#   "scripts": {
#     "test": "echo \"Error: no test specified\" && exit 1"
#   },
#   "author": "Markus Reichl",
#   "license": "MIT"
# }

% curl localhost:8009/nosuch.file
# File not found
```