High-level webszerverek: express

Webprogramozás – 7. előadás

Sulyok Csaba
csaba.sulyok@gmail.com

1. rész

Alacsonyszintű webszerver problémái

- Nehézkes a webszerverek implementálása
- Külön oda kell figyelni a különböző események lekezelésére
- Létezik sokkal egyszerűbb megoldás külső modulok formájában
- pl. `Express` - a *"Fast, unopinionated, minimalist web framework for node.js."*
- `npm install --save express`
- `import express from 'express';`

Express: Hello World!

```
import express from 'express';

// alkalmazás elkészítése
const app = express();

// GET hívások a "/hello" útvonalra itt landolnak
app.get('/hello', (req, res) => {
  console.log('Received a new request');
  res.send('And hello to you too.');
});

app.listen(8080, () => { console.log('Server listening...'); });
```

- **Példa:** 3-nodejs/simpleserver/simpleserver_express.js

Express: statikus állományok

- **Példa:** `3-nodejs/staticserver/staticserver_express.js`

```
import express from 'express';
import path from 'path';

// a mappa ahonnan statikus tartalmat szolgálunk
// process.cwd() - globális változó, az aktuális katalógusra mutat a szerveren
// SOSE a gyökeret tegyünk publikussá
const staticDir = path.join(process.cwd(), 'static');

// inicializáljuk az express alkalmazást
const app = express();

// express static middleware: statikus állományokat szolgál fel
app.use(express.static(staticDir));

app.listen(8080, () => { console.log('Server listening...'); });
```

Express: logger middleware

- több eseménykezelő is beszúrható az express hívásaiba
- pl. `morgan` – naplózási (logger) függőség
- npm függőség szükséges a `package.json`-ban
- adaptált példa:

```
import express from 'express';
import { join } from 'path';
import morgan from 'morgan';

// a mappa ahonnan statikus tartalmat szolgálunk
// process.cwd() - globális változó, az aktuális katalógusra mutat a szerveren
// SOSE a gyökeret tegyünk publikussá
const staticDir = join(process.cwd(), 'static');

// inicializáljuk az express alkalmazást
const app = express();

// morgan middleware: loggolja a beérkezett hívásokat
app.use(morgan('tiny'));
// express static middleware: statikus állományokat szolgál fel
app.use(express.static(staticDir));

app.listen(8080, () => { console.log('Server listening...'); });
```

Express routing

- a létrehozott alkalmazáshoz hozzárendelhetünk kezelő callbackeket a HTTP metódusnevekből derivált metódusokkal, vagy az `all` metódussal:

```
app.get('/myroute', (req, res) => { ... });  // GET /myroute esetén
app.post('/myroute', (req, res) => { ... }); // POST /myroute esetén
app.all('/myroute', (req, res) => { ... });  // bármilyen metódus esetén a route-ra
```

- reguláris kifejezések is alkalmazhatóak:

```
app.get(/.*fly$/, (req, res) => { ... }); // /butterfly, /dragonfly, etc.
```

- dinamikus változókat helyezhetünk el az útvonalba:

```
app.get('/myroute/:myparam', (req, res) => {
  const myparam = req.params.myparam;
  console.log(myparam); // // GET /myroute/value  =>  kimenet: 'value'
});
```

- a `use` használatával beszúrhatunk lekezelőket, melyek érvényesek minden metódusra és bármely **al**útvonalra is - ebben az esetben az útvonal-paraméter kihagyásával az minden hívásra érvényes lesz:

```
app.use('/myroute', (req, res) => { ... });  // GET /myroute/subroute esetén is
app.use((req, res) => { ... }); // minden hívást kezel
```

Express middleware

- egy bizonyos útvonalra több callback függvény is érvényes lehet, pl. a `GET /myroute`-ra reagálhatnak a következők:

```
app.get('/myroute', (req, res) => { ... });
app.all('/myroute', (req, res) => { ... });
app.use('/', (req, res) => { ... });
app.get(/rout/, (req, res) => { ... });
```

- a találó callback függvények közül az első deklarált van meghívva a `request`, `response` és `next` paraméterekkel, ahol a `next` a következő függvényre mutat az érvényes láncban
- ha egy callback függvény célja nem az, hogy végleges választ építsen fel, hanem szűrje, feldolgozza a hívásokat, majd továbbengedje a hívásláncot a `next` függvény segítségével, azt egy **middleware**-nek nevezzük - általában a `use` metódussal renduljük a feldolgozási lánchoz
- a híváslánc megszakad egy valid HTTP válasz visszatérítésekor (a `response end` vagy `send` metódusa) vagy ha egy `next` függvény nincs megfelelően meghívva a láncban
- *vigyázat:* ha egy válasz felépítését mellőzzük és a `next` függvényt sem hívjuk meg, **nem tér vissza válasz**

Express middleware példa

Példa: `3-nodejs/express_middleware` – middleware, amely kinaplóz minden hívást, mielőtt továbbengedi egy statikus feldolgozónak

```
import express from 'express';
import { join } from 'path';

const staticDir = join(process.cwd(), 'static');

const app = express();

// loggoló middleware - minden híváskor kikötünk itt
app.use((req, resp, next) => {
  const date = new Date();
  console.log(`${date} ${req.method} ${req.url}`);
  // továbbengedjük a hívási láncot
  next();
});

app.use(express.static(staticDir));

app.listen(8080, () => {
  console.log('Server listening on http://localhost:8080/ ...');
});
```

Express Router

- az express `app` egy router, mely a gyökérre érkező hívásokat kezeli
- ezen kívül készíthetünk egyedülálló `express.Router` példányokat, melyeket hozzárendelhetünk az `app`-hoz vagy más router példányhoz
- így moduláris és jól átlátható szerkezetet alakíthatunk ki
- az `express.Router` példányokat konvencionális kivesszük a főállományból külső modulba, amelyek a router példányt exportálják
- ugyanazon bekötő metódusokat használhatjuk, mint a gyökérpéldányon (`get`, `post`, `<othermethod>`, `all`, `use`), de globális metódusok (pl. `listen`, `render`) nem elérhetőek

Express Router példa

Példa: 3-nodejs/express_router

routes/birds.js:

```
import { Router } from 'express';

// felépítünk egy moduláris routert
const router = Router();

// csak ezen routerhez tartozó middleware
router.use((req, res, next) => {
  console.log(`${req.method} ${req.url}`);
  next();
});
// homepage
router.get('/', (req, res) => {
  res.send('Birds home page');
});
// about
router.get('/about', (req, res) => {
  res.send('About birds');
});

module.exports = router;
```

index.js:

```
import express from 'express';
import birds from './routes/birds.js';

const app = express();

/*
Bizonyos URL-re kötött router
A path-ek konkatenálódnak fastruktúra szerint
Hívások mehetnek a következőkre:
 - /birds/
 - /birds/about
*/
app.use('/birds', birds);

app.listen(8080);
```

2. rész

Formfeldolgozás

HTML5 űrlapok (formok) - ismétlés

```
<form action="/submit_form" method="POST">
  <p>Név: <input type="text" name="name" value="Alapértelmezett érték" /></p>
  <p>Jelszó: <input type="password" name="password" /></p>
  <p>Születési dátum: <input type="date" name="birthdate" /></p>
  <p>Nem:
    <select name="gender">
      <option value="male">Férfi</option>
      <option value="female" selected>Nő</option>
      <option value="other">Más</option>
    </select>
  </p>
  <p>HTML tudás: <input type="checkbox" name="knowshtml"></p>
  <p><input type="submit" value="Leadom!" /></p>
</form>
```

- `method`: HTTP kérés metódusa (`GET` vagy `POST`)
- `action`: a cél URL, ahova a form adatok el lesznek küldve (*lehet relatív is*)
- `onsubmit`: esemény, amely a formadatok elküldése előtt hívódik meg (ha `false`-ot térít vissza, a form nem lesz elküldve)
- csak a `name` attributummal ellátott mezők lesznek elküldve

HTML5 űrlapok (formok)

- A böngésző a HTML kérésbe behelyezi az adatokat - a formátumuk a form `enctype` attribútumától függ
- Ennek alapértelmezett értéke `application/x-www-form-urlencoded` - nyers URL paraméter formázás
- Az előbbi példa a következő kéréstörzset eredményezi:

```
name=My%20Name&password=mypass&birthdate=1980-04-01&gender=male&knowshtml=on
```

- `method="GET"` esetén ez látszik a böngésző fejlécében (tehát érzékeny információt **sose** küldjünk így)
- `method="POST"` esetén ugyanez a string bekerül a törzsbe
- mindkét esetben az `express`-nek dekódolnia kell információt, ehhez használjuk az `express.urlencoded` middleware-t
- Postman replikációért válasszuk az `x-www-form-urlencoded` opciót a HTTP hívás törzsének

HTML5 űrlapok példa

Példa: `3-nodejs/formhandling`

```
// standard kérésfeldolgozással kapjuk a body tartalmát
app.use(express.urlencoded({ extended: true }));

// formfeldolgozás
app.post('/submit_form', (request, response) => {
  const birthDate = new Date(request.body.birthdate);
  const knowsHtml = Boolean(request.body.knowshtml);

  const respBody = `A szerver sikeresen megkapta a következő információt:
    név: ${request.body.name}
    jelszó: ${request.body.password} (sosem szabad ezt így kiírni :))
    születési dátum: ${birthDate}
    nem: ${request.body.gender}
    tud HTML-t: ${knowsHtml}
  `;
  console.log(respBody);

  response.set('Content-Type', 'text/plain;charset=utf-8');
  response.end(respBody);
});
```

HTML5 űrlapok: állományok feltöltése

```
<form action="/upload_file" method="POST" enctype="multipart/form-data">
  <p>Állomány: <input type="file" name="myfile" /></p>
  <p>Privát: <input type="checkbox" name="private" /></p>
  <p><input type="submit" value="Leadom!" /></p>
</form>
```

- az alapértelmezett `enctype` (URL-encoded) nem alkalmas file-ok feltöltésére, ezért az alternatív `multipart/form-data`-t alkalmazzuk
- csak `POST` metódus esetén alkalmazható
- megengedi a `file` típusú `input` elem jelenlétét
- az `express.urlencoded` middleware nem alkalmas ezen formok feldolgozására
- ezért alkalmazzuk a `formidable` csomagot, amely elvégzi a feldolgozást
- express middleware bekötéséért alkalmazzuk az `express-formidable` összekötő csomagot

Állományok feltöltése: példa

Példa: 3-nodejs/fileupload

```
import express from 'express';
import { existsSync, mkdirSync } from 'fs';
import { join } from 'path';
import eformidable from 'express-formidable';

const app = express();
const uploadDir = join(process.cwd(), 'uploadDir');

// feltöltési mappa elkészítése
if (!existsSync(uploadDir)) {
  mkdirSync(uploadDir);
}

// a static mappából adjuk a HTML állományokat
app.use(express.static(join(process.cwd(),
  'static')));

// formidable-lel dolgozzuk fel a kéréseket
app.use(eformidable({ uploadDir }));
```

```
// formfeldolgozás
app.post('/upload_file', (request, response) => {
  // az állományok a request.files-ban lesznek
  const fileHandler = request.files.myfile;
  // más mezők a request.fields-ben
  const privateFile = Boolean(request.fields.private);

  const respBody = `Feltöltés érkezett:
    állománynév: ${fileHandler.name}
    név a szerveren: ${fileHandler.path}
    privát: ${privateFile}
  `;

  console.log(respBody);
  response.set('Content-Type',
    'text/plain;charset=utf-8');
  response.end(respBody);
});

app.listen(8080, () => {
  console.log('Server listening...');
});
```