node : javascript côté serveur – Express

javascript et son écosystème

Licence mention Informatique
Université Lille – Sciences et Technologies

Serveur web minimal

basic-server/minimal

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

const server =  http.createServer (
   function(request, response) {
     response.writeHead(200, { "Content-Type": "text/html" });
     response.write('<h1>First minimal node server</h1>');
     response.write('<p>I am alive</p>');
     response.end('bye');
  }
);

server.listen(8080);
```

- http://127.0.0.1:8080/

nodemon

- npm install nodemon --global
- nodemon
  redémarrage du serveur à chaque modification

basic-server/second

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

const server = http.createServer (
  function(request, response) {
    const path =  url.parse(request.url).pathname;
    const params =  querystring.parse(url.parse(request.url).query);
    let name = "unknown";
    if ('name' in params) {
      name =  params['name'];
    }

    response.writeHead(200, { "Content-Type": "text/html" });
    response.write('<h1>Second node server</h1>');
    response.write('<p style="...">url is '+url+' path is '+path+'</p>');
    response.write('<p>hello '+name+'</p>');
    response.end();
  }
);

server.listen(8080);
```

- http://127.0.0.1:8080/client.html?name=timoleon

express

un *framework* web pour Node.js

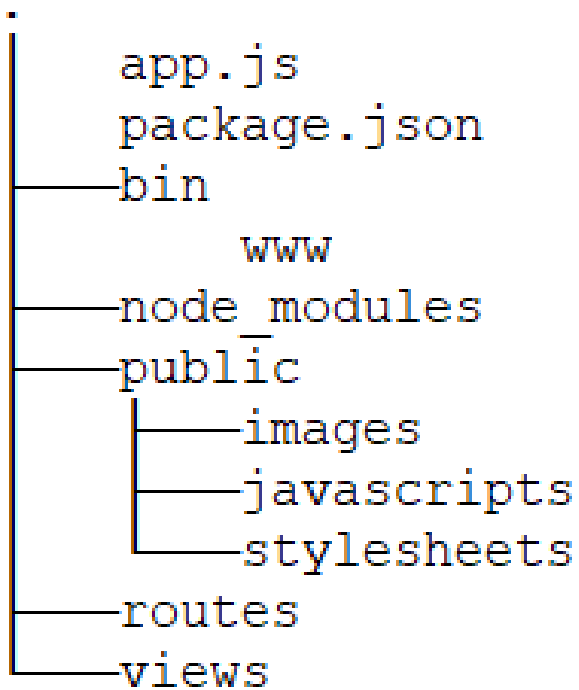
express

- permet de fixer les propriétés classiques d'un serveur
- facilite la gestion des routes
- s'appuie sur des *middlewares* qui peuvent s'insérer dans le flux de gestion des requêtes
- intègre des moteurs de rendu des vues

installation

- utilisation de l'outil `express-generator`
  `npm install express-generator -g`
- création du projet
  `express --view=pug <app_name>`
  utilisation de *pug* comme moteur de vues
  ↪ crée une structure par défaut pour les projets `express`
- installation des dépendances dans dossier *app_name*
  `npm install`
- `npm start` → `http://127.0.0.1:3000`

structure du projet



express v0

bin/www

express v0

- dans package.json

```
"scripts": {
  "start": "node ./bin/www"
}
```

- mise en place du serveur http Node.js

```
// dans /bin/www
var app = require('../app');
var http = require('http');
    (...)
app.set('port', port);
    (...)
var server = http.createServer(app);
    (...)
server.listen(port);
server.on('error', onError);
    (...)
function(onError) { ...
```

app.js

- structure générale

```
// dans /app.js
const express = require('express');
(...)
const app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');

// mount middlewares
app.use(...);
app.use(...);
...
```

- `__dirname` est une variable `Node.js` représentant le dossier du module courant
- `app.use(...)` insère un *middleware* dans le flux de traitement de la requête

app.use

app.use([path,] callback [, callback...])

- **path** précise le chemin auquel s'applique le middleware
  - si absent, le middleware s'applique à tous les chemins
  - peut être une expression régulière
- **callback** peut être une « **fonction middleware** »

autres possibilités : *Express API*

- fonction middleware : `function(req, res, next)`
  - `req` l'objet requête HTTP, de type `Request`
  - `res` l'objet réponse HTTP, de type `Response`
  - `next` fonction callback pour chainer les middlewares

exemples

- ```
  app.use(express.static(path.join(__dirname, 'public')));
  ```

  `express.static` est un middleware pour gérer de manière statique les fichers (css, png, etc.) placés sous le dossier précisé (ici `public`)
  http://127.0.0.1:3000/images/timoleon.jpg

  express v0

- ```
  app.use(express.json());
  app.use(express.urlencoded({ extended: false }));
  ```

  analyse le corps de la requête (au format json ou url-encoded) et le restitue dans `req.body`

- ```
  app.use(cookieParser());
  ```

  analyse l'entête `Cookie` de la requête et alimente `req.cookies`
  ```

- les middlewares sont exécutés séquentiellement
- pour que l'on passe au suivant il faut qu'un middleware appelle `next()`
- exemples (triviaux) de middlewares
  - ajout d'un middleware trivial pour la route `/first`

```
// dans /app.js
app.use('/first',
  (req, res, next) => {
          res.writeHead(200, {"Content-Type": "text/html"});
          res.write('<h1>first middleware</h1>');
          res.write('<p>I am alive and using route /first</p>');
          res.end('bye...');
  });
```

express v1

  - on peut « chainer » des middlewares avec `next()`

```
// dans /app.js
app.use(
  function(req, res, next) {
    req.body.witness = 'Timoleon was here';
    next();
  }
);
```

express v1.1

routeurs

un routeur

- est créé par la méthode `Router()` de l'objet `express`
- est un middleware dédié à la gestion des routes
- se comporte comme une "*mini application*"
  dispose d'une méthode `use()`
- gère les routes pour les différentes requêtes HTTP (GET, POST, ...)
  ↪ méthodes `.get()`, `.post()`, etc.

bonne pratique

- définir chaque routeurs dans un module à part et exporter ce routeur
- l'importer et le déclarer, avec sa « route racine », comme middleware dans l'application

  Node.js utilise la syntaxe *common.js* pour gérer les modules : `require`/`module.exports`. La syntaxe ES6 est encore expérimentale.

```
// dans app.js
var booksRouter = require('./routes/books');
...
app.use('/books', booksRouter);
```

- NB : dans le routeur, les chemins sont relatifs à la route racine déclarée pour le middleware
- cf. dossier `routes` généré par Express

express v2

ajout de middleware au routeur

express v2.1

```
// dans /routers/books.js
const books = require('../data/books'); // to simulate data acquisition
const router = express.Router();

const allBooks;
const bestBook;

router.use(
(req, res, next) => {
    bestBook = books[0];
    allBooks = Array.of(...books);
    allBooks.sort( (book1, book2) => book1.author.localeCompare(book2.author) );
    next();
  }
)

router.get('/',... );
router.get('/best',...);

module.exports = router;
```

chemins des routes

- routes “joker” ’?’, ’+’, ’*’

```
router.get('/root/ti?mo*on',   ... );
```

/root/timoleon, /root/tiXmoLLEEon, /root/tiXmoon, etc.

- routes patterns ou expressions régulières

```
router.get(/[bB]e(st|ST)/, ...);
```

/best, /Best, /beST, /BeST

```
router.get(/.*[bB]est.*/, ...);
```

/best, /Best, /theBestOne, /it/is/reallythebest/one , etc.

express v2.2

chemins des routes

- routes paramétrées

```
// dans /routers/books.js
router.get('/details/:bookId',
  (req, res, next) => res.render('bookdetail',
                                 { title: 'Book detail',
                                   id: req.params.bookId,
                                   book:  books[req.params.bookId - 1]
                                 })
);
```

  - pas forcément en fin de chemin : `/details/:bookId/other`
  - plusieurs paramètres possibles : `/details/:bookId/:nextId`

express v2.2

bonne pratique

séparation des préoccupations

- séparer la gestion des routes de leur logique de traitement
  - définir un **contrôleur** pour chaque route
    dans un dossier `controllers`
  - le contrôleur définit et exporte les fonctions de traitement
  - le routeur importe le contrôleur pour en utiliser les fonctions

express v2.3

définition de la logique des routes dans le contrôleur

```
    // dans /controllers/books.js
let prepare = (req, res, next) => { ... };
let list = (req, res) => res.render( ...  ); // controller function for '/'
let best = (req, res) => res.render( ... );
let details = (req, res) => res.render( ... );
    // export controller functions
module.exports.prepare = prepare;
module.exports.list = list;
module.exports.best = best;
module.exports.details = details;
```

utilisation par le routeur

```
  // dans /routers/books.js
const booksController = require('../controllers/books');

  // link controllers to route paths
router.use( booksController.prepare );
router.get('/', booksController.list );
router.get(/[bB]est/, booksController.best );
router.get('/details/:bookId', booksController.details );
```

express v2.3

classes ES6

utilisation des classes ES6 pour définir les contrôleurs :

```
    // dans /controllers/books.js
class BookController {
  constructor() {
    this.books = require('../data/books');
    ...
    // bind method to this
    this.prepare = this.prepare.bind(this);
    this.list = this.list.bind(this);
    ...
  }
  // controller methods (used as middlewares)
  prepare(req, res, next) { ... }
  list(req, res) { ... }
  best(req, res) { ... }
  details(req, res) { ... }
}
// export controller instance
module.exports = new BookController();
```

express v2.3.5

principales méthodes de `Response`

- `res.download()` Prompt a file to be downloaded.
- `res.end()` End the response process.
- `res.json()` Send a JSON response.
- `res.redirect()` Redirect a request.
- `res.render()` Render a view template.
- `res.send()` Send a response of various types.
- `res.sendFile()` Send a file as an octet stream.
- `res.status()` Set the response status code

réponse json et autres

- `res.status()` fixe la code du statut de la réponse, renvoie une réponse (« chainable »)
- `res.json()` envoie une réponse au format JSON

```
// dans /controllers/books.js
var details =
    (req,res) => res.status(200).json( books[req.params.bookId - 1] );
```

- `res.sendFile()` voir dans `/controllers/example.js`
- `res.download()` voir dans `/controllers/example.js`
- `render` utilisation d'un moteur de vue

express v2.4

res.render

express v2.5

```
// dans /controllers/index.js
module.exports.home =
     (req, res) => res.render('index', { title:  'Express' });
```

`res.render(view, locals)`

- *view* : la vue à construire
- *locals* : un objet qui définit des propriétés locales à la vue

```
//dans /app.js
      // view engine setup
app.set('views', path.join(__dirname,'views'));
app.set('view engine', 'pug');
```

- bonne pratique : un dossier pour les vues `/views`
- **pug** est un **moteur de templates** pour code HTML
- le processus de pug génère du code HTML pour des templates écrits dans la syntaxe de pug

pug : syntaxe html allégée

```
// dans /views/about.pug
doctype html

html
  head
    title about page
    link(rel="stylesheet", href="/stylesheets/style.css")
                                                  // attributs entre ()
  body
                                                  // indentation pour "emboitement"
    h1 about page                                 // balise html avec contenu
    p.text Introduction to Express                // avec une classe CSS
    .example Licence 3 Informatique - option      // div implicite
      em Javascript et son écosystème
    #otherexample                                 // id avec div implicite
      | voir                                   // texte simple sur plusieurs lignes
      |
      a(href="http://portail.fil.univ-lille1.fr/ls6/js") sur le portail
```

template

- notion de **block** que les *templates enfants* peuvent (ou non) remplacer

```
///views/layout.pug
doctype html

html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    block content

  footer
    block footblock
      p Licence 3 - #[em Javascript et son écosystème]
```

- NB : *tag interpolation* : `#[em ...]`

héritage

- héritage partiel

```
// dans /views/index.pug
extends layout

   block content
      h1= title
      p Welcome to #{title}
```

- héritage avec extension (prepend existe aussi)

```
//  dans /views/booklayout.pug
extends layout
 (...)
 block content
    h1 about books block bookcontent

 append footblock
    p you are in #[em book zone]
```

interpolation

utilisation des valeurs passées dans le paramètre `locals` de `render`

```
//  dans /controllers/index.js
module.exports.home =
   (req, res) => res.render('index', { title:  'Express' });
```

```
// dans /views/index.pug
extends layout

  block content
    h1= title
    p Welcome to #{title}
```

itération

structure itérative pour aider à la définition des templates

```
// dans /controllers/books.js
var list =
  (req, res) => res.render('books', { title: '...', books :  allBooks });
```

```
// dans /views/books.pug
extends booklayout

block bookcontent
  h1= title
  p Welcome to #{title.toLowerCase()}
  table.booklist
    each book in books
      tr.book
        td.author= book.author
        td.title= book.title
```

code javascript

préfixer les lignes de code par « - »

```
// dans /views/users.pug
extends layout

block content
  - var users = ['Tim Oleon', 'Timo Léon', 'Ti Moléon'];

  h1= title
  p Welcome to #{title}
  p here comes the user list...
  ul
    - for (let i = 0 ; i < users.length ; i++)
      li user-#{i} = #{users[i]}
```

mixin

les mixins sont des blocs réutilisables et paramétrables « mini composants »

- ```
  // dans /views/booklayout.pug
  ...
  mixin createCover(imgSrc)
    .cover
      img(src=imgSrc)
  ...
  ```

- utilisation de + pour « appeler » le mixin

```
// dans /views/bookdetail.pug
block bookcontent
  h1= title
  ...
  +createCover(‘$book.cover‘)

// dans /views/betsbook.pug
block bookcontent
  ...
  +createCover(‘$book.cover‘)
```
```