Node.js i TypeScript - jak zacząć?

Adam Stolcenburg

22 listopada 2017

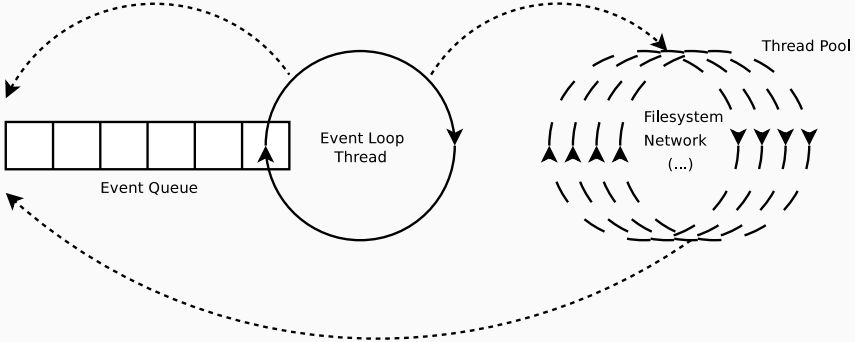
Akademia ADB

Node.js

Czym jest node.js?

- https://nodejs.org
- licencja MIT - https://github.com/nodejs/node
- działa na Linux, macOS, Solaris, FreeBSD, OpenBSD, Microsoft Windows
- komponenty składowe:
  - V8 - silnik JavaScript wykorzystywany przez Chrome
  - libuv - wieloplatformowa biblioteka ze wsparciem do asynchronicznych operacji wejścia-wyjścia
  - http-parser
  - c-ares
  - OpenSSL
  - zlib
- ponad 40 wbudowanych i 475,000 zewnętrznych pakietów
- jednowątkowy, sterowany zdarzeniami model programowania

Jednowątkowy, sterowany zdarzeniami model programowania



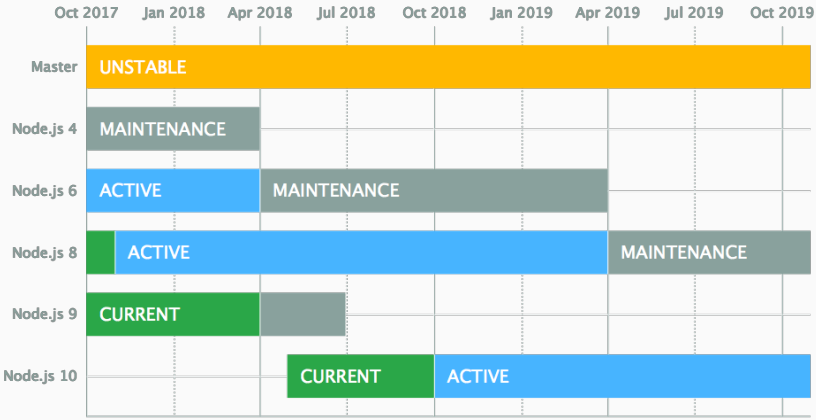
Zastosowanie

- oprogramowanie serwerowe
  - JavaScript po stronie serwera
  - zdolny do obsłużenia wielu tysięcy równoległych połączeń
- urządzenia wbudowane
  - brak konieczności kompilacji
  - ten sam kod możliwy do uruchomienia zarówno na PC jak i na urządzeniu
- skrypty

Wydajność

- zgrubsza (https://benchmarksgame.alioth.debian.org/u64q/javascript.html)
  - wolniejszy niż C++
  - wolniejszy niż Java
  - szybszy niż Python
- w razie potrzeby można użyć C++ - https://nodejs.org/api/addons.html

Wersje



https://github.com/nodejs/Release#release-schedule

Instalacja

- źródła oraz binarki do pobrania z oficjalnej strony:
  https://nodejs.org/en/download/
- instalacja za pomocą managera pakietów:
  https://nodejs.org/en/download/package-manager/
  - Debian i Ubuntu:

```
curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
sudo apt-get install -y nodejs
```

- NVM (Node Version Manager) - https://github.com/creationix/nvm

Uruchamianie kodu JavaScript

- parametr uruchomienia

```
node hello.js
```

- REPL - Read-Eval-Print-Loop

```
$ node
> console.log(‘Hello ${process.env.USER}!‘);
```

- skrypt

```
#!/usr/bin/env node
console.log(‘Hello ${process.env.USER}!‘);
```

NPM - Node Package Manager

- menedżer pakietów JavaScript współdziałający z rejestrem https://www.npmjs.com/
- instalowanie, zarządzanie zależnościami

```
npm install [package[@version]]
npm install -g <package[@version]>
npm install --save <package[@version]>
npm install --save-dev <package[@version]>
```

- publikowanie
- uruchamianie

```
npm start
npm run <script>
```

- tworzenie nowych pakietów/aplikacji

```
npm init
```

package.json

```
{
  "name": "my-app",
  "version": "1.0.0", "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "babel": "babel index.js"
  },
  "author": "Me", "license": "MIT",
  "devDependencies": {
    "babel-cli": "^6.26.0"
  },
  "dependencies": {
    "node-fetch": "1.7.3"
  }
}
```

Modularność

- CommonJS (CJS), Asynchronous Module Design (AMD), ECMAScript 2015 (ES2015)
- rodzaje modułów
  - wbudowany pakiet - https://nodejs.org/dist/latest-v8.x/docs/api/
  - zewnętrzny pakiet - https://www.npmjs.com/
  - plik
  - katalog z plikiem index.js
- ładowanie modułów w node.js (CommonJS)

```
const os = require('os');
const local = require('./local');
```

Modularność wewnątrz projektu

cpus.js

```
const os = require('os');
function print() {
    console.log(`${JSON.stringify(os.cpus(), null, 2)}`);
}
console.log('Hello from cpus.js!');
module.exports.print = print;
module.exports.get = os.cpus;
```

index.js

```
const cpus = require('./cpus');
cpus.print();
console.log(`${JSON.stringify(require('./cpus').get())}`);
```

Synchroniczny model programowania

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

try {
    fs.writeFileSync('./out.txt', 'zapisany tekst');
    const text = fs.readFileSync('./out.txt', 'utf8');
    console.log(text);
    fs.unlinkSync('./out.txt');
} catch (err) {
    console.log(`${err}`);
}
```

Asynchroniczny model programowania - callback

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

fs.writeFile('./out.txt', 'zapisany tekst', function (err) {
    if (err) return console.log(`${err}`);

    fs.readFile('./out.txt', 'utf8', function (err, text) {
        if (err) return console.log(`${err}`);

        console.log(text);
        fs.unlink('./out.txt', function (err) {
            if (err) return console.log(`${err}`);
        });
    });
});
```

Asynchroniczny model programowania - promise

```
const promisify = require('util').promisify;
const fs = require('fs');

promisify(fs.writeFile)('./out.txt', 'zapisany tekst')
    .then(() => {
        return promisify(fs.readFile)('./out.txt', 'utf8');
    })
    .then(text => {
        console.log(text);
        return promisify(fs.unlink)('./out.txt');
    })
    .catch(err => {
        console.log(`${err}`);
    });
```

Asynchroniczny model programowania - async/await

```
const promisify = require('util').promisify;
const fs = require('fs');

async function print() {
    try {
        await promisify(fs.writeFile)('./out.txt', 'zapisany tekst');
        let text = await promisify(fs.readFile)('./out.txt', 'utf8');
        console.log(text);
        await promisify(fs.unlink)('./out.txt');
    } catch (err) {
        console.log(`${err}`);
    }
}

print();
```

TypeScript

JavaScript

ECMAScript - język ustandaryzowany w specyfikacji ECMA-262 i ISO/IEC 16262

- ECMAScript 3 (ES3) - 1999
- ECMAScript 5 (ES5) - 2009/2011
  - http://kangax.github.io/compat-table/es5/
- ECMAScript 2015 (ES2015, ES6)
  - klasy i dziedziczenie
  - funkcje strzałkowe (ang. arrow functions)
  - const i let
  - obietnice (ang. promise objects)
  - moduły
  - http://kangax.github.io/compat-table/es6/
- ECMAScript 2017 (ES2017)
  - funkcje asynchroniczne (ang. async functions)
  - http://kangax.github.io/compat-table/es2016plus/

TypeScript

- napisany i wspierany przez Microsoft - https://www.typescriptlang.org/
- licencja Apache License 2.0 - https://github.com/Microsoft/TypeScript
- nadzbiór języka JavaScript
  - wsparcie dla kontroli typów
  - interfejsy
  - programowanie generyczne
  - tuple
  - przestrzenie nazw

Dziedziczenie w TypeScript

```
class Base {
    protected value = 5;
    protected mult: number;
    constructor(private inValue: number, offset: number) {
        this.mult = this.value * this.inValue + offset;
    }
    protected print(): void {
        console.log(‘Base value ${this.value}‘);
    }
}
class Derived extends Base {
    public printAll(): void {
        this.print();
        console.log(‘Derived value ${this.value * this.mult}‘);
    }
}
new Derived(10, 20).printAll();
```

Dziedziczenie w ES2015

```
class Base {
    constructor(inValue, offset) {
        this.inValue = inValue;
        this.value = 5;
        this.mult = this.value * this.inValue + offset;
    }
    print() {
        console.log(`Base value ${this.value}`);
    }
}
class Derived extends Base {
    printAll() {
        this.print();
        console.log(`Derived value ${this.value * this.mult}`);
    }
}
new Derived(10, 20).printAll();
```

Dziedziczenie w ES5 i

```
var Base = (function () {
    function Base(inValue, offset) {
        this.inValue = inValue;
        this.value = 5;
        this.mult = this.value * this.inValue + offset;
    }
    Base.prototype.print = function () {
        console.log("Base value " + this.value);
    };
    return Base;
}());
var Derived = (function (_super) {
    __extends(Derived, _super);
    function Derived() {
```

Dziedziczenie w ES5 ii

```
        return _super !== null && _super.apply(this, arguments) || this;
    }
    Derived.prototype.printAll = function () {
        this.print();
        console.log("Derived value " + this.value * this.mult);
    };
    return Derived;
}(Base));
new Derived(10, 20).printAll();
```

Modularność - TypeScript (ES2015)

cpus.ts

```
import * as os from 'os';
export function print(): void {
    console.log(`${JSON.stringify(os.cpus(), null, 2)}`);
}
console.log('Hello from cpus.ts!');
export { cpus as get } from 'os';
console.log('os.arch()');
```

index.ts

```
import * as cpus from './cpus';
cpus.print();
console.log(`${JSON.stringify(cpus.get())}`);
```

Kompilator

- napisany w TypeScript
- dostępny jako pakiet npm
  - instalacja jako samodzielna aplikacja

    ```
    npm install -g typescript
    ```

  - jako zależność projektu

    ```
    npm install --save-dev typescript
    ```

- transpilacja

  ```
  tsc index.ts
  ```

Ważne opcje kompilatora

- `-h, -help` – wyświetla pomoc
- `-all` – wyświetla szczegółową pomoc
- `-init` – tworzy plik tsconfig.json
- `-p FILE OR DIRECTORY, -project FILE OR DIRECTORY` – pozwala wskazać katalog z którego ma być brana konfiguracja projektu
- `-w, -watch` – monitoruje i automatycznie transpiluje pliki `.ts` do `.js`
- `-t VERSION, -target VERSION` – pozwala zdfiniować docelową wersję ECMAScript: `ES3` (domyślny), `ES5`, `ES2015`, `ES2016`, `ES2017`, `ESNEXT`
- `-m KIND, -module KIND` – pozwala zdefiniować rodziaj modularności, jeden z: `commonjs`, `amd`, `system`, `umd`, `es2015`, `ESNext`
- `-d, -declaration` – generuje dodatkowo pliki `.d.ts`
- `-sourceMap` – generuje dodatkowo pliki `.map`
- `-strict` – bardziej rygorystyczna weryfikacja kodu

tsconfig.json

```
{
  "compilerOptions": {
    "target": "esnext",
    "module": "commonjs",
    "sourceMap": true,
    "outDir": "./out",
    "rootDir": "./src",
    "strict": true
  }
}
```

Teraz wystarczy uruchomić:

```
tsc -p .
```

Współpraca z pakietami node.js

- informacje o typach modułów wbudowanych w node.js dostarcza pakiet `@types/node`
- pakiety dostarczające pliki `.d.ts` mogą być używane bezpośrednio
- popularne pakiety posiadają odpowiadające im pakiety z typami w zakresie `@types`, na przykład dla pakietu `node-fetch` definicja typu dostarczona jest za pomocą pakietu `@types/node-fetch`
- pozostałe pakiety można użyć:
  - pisząc własną definicje typów - http://www.typescriptlang.org/docs/handbook/declaration-files/introduction.html
  - bez informacji o typach

package.json współpracujący z tsconfig.json

```
{
  "name": "my-app",
  "version": "1.0.0",
  "description": "My application",
  "license": "MIT",
  "main": "./out/index.js",
  "scripts": {
    "start": "node ./out/index.js",
    "build": "tsc -p .",
    "release": "tsc -p . --sourceMap false --outDir ./release"
  },
  "author": "Me",
  "devDependencies": {
    "@types/node": "^8.0.53",
    "typescript": "^2.6.1"
  }
}
```

Visual Studio Code

Visual Studio Code

- napisany i wspierany przez Microsoft - https://code.visualstudio.com/
- źródła na licencji MIT - https://github.com/Microsoft/vscode
- działa na Linux, Microsoft Windows i macOS
- doskonałe wsparcie dla TypeScript
- napisany w TypeScript i JavaScript, korzysta z node.js
- do ściągnięcia z https://code.visualstudio.com/

Visual Studio Code - wsparcie dla TypeScript

- inteligentne uzupełnianie kodu
- sprawdzanie poprawności w czasie pisania
- transpilacja z poziomu edytora
- uruchamianie i debugowanie
- formatowanie
- refaktoryzacja
- integracja z GIT

Dziękuję

Warto przeczytać

Node.js Design Patterns - Second Edition, Mario Casciaro, Luciano Mammino

https://www.typescriptlang.org/docs/home.html