Tous les bundlers au monde

Introduction

JavaScript n’est pas modulaire.

Il le sera peut-être un jour.

En attendant, on doit coder.

On a besoin de modularité.

Parlons des bundlers.

Vue d’ensemble

Les bundlers simulent la modularité en JavaScript.

Il en existe beaucoup.

D’abord, les task runners.

Ensuite, les asset builders.

Enfin, les module bundlers.

Les task runners

Les task runners sont des outils capables de lancer des tâches prédéfinies.

Installation

Grunt

```
npm install -D \
  babel-preset-es2015 \
  grunt \
  grunt-babel \
  grunt-cli \
  grunt-contrib-concat
```

Gulp

```
npm install -D \
  babel-preset-es2015 \
  gulp \
  gulp-babel \
  gulp-concat
```

Exécution

Grunt

grunt build

Gulp

gulp build

Configuration

Grunt

Gruntfile.js

```
module.exports = (grunt) => {
  const babel = {
    build: {
      dest: 'dist/bundle.js',
      src: 'dist/bundle.js.tmp'
    }
  }
```

```
const concat = {
  build: {
    dest: 'dist/bundle.js.tmp',
    src: 'app/**/*.js'
  }
}
```

```
  grunt.initConfig({babel, concat})

  grunt.loadNpmTasks('grunt-babel')
  grunt.loadNpmTasks(
    'grunt-contrib-concat')

  grunt.registerTask('build',
    ['concat', 'babel'])
}
```

.babelrc

```
{
  "presets": ["es2015"]
}
```

Gulp

gulpfile.js

```
const gulp = require('gulp')
const babel = require('gulp-babel')
const concat = require('gulp-concat')
```

```
gulp.task('build', () => {
  return gulp.src('app/**/*.js')
    .pipe(concat('bundle.js'))
    .pipe(babel())
    .pipe(gulp.dest('dist'))
})
```

.babelrc

```
{
  "presets": ["es2015"]
}
```

Application

index.js

hello()

banner.js

```
const name = 'Devfest Lille'
const year = 2017

function hello () {
  console.log(`Bonjour ${name}.`)
}
```

Compilation

Le bundle est la concaténation de tous vos fichiers sans isolation.

Évaluation

Les task runners sont faciles à mettre en place.

Les task runners sont faciles à utiliser.

Les task runners sont compliqués à maintenir.

Les task runners ratent le support de la modularité.

Recommandation

Les task runners sont adaptés pour apprendre.

Les asset builders

Les asset builders sont des outils compatibles avec ECMAScript 5.

Installation

Brunch

```
npm install -D \
  babel-brunch \
  babel-preset-es2015 \
  brunch
```

Broccoli

```
npm install -D \
  babel-preset-es2015 \
  broccoli \
  broccoli-babel-transpiler \
  broccoli-cli \
  broccoli-concat
```

Exécution

Brunch

brunch build

Broccoli

broccoli build dist

Configuration

Brunch

brunch-config.js

```
exports.files = {
  javascripts: {
    joinTo: 'bundle.js'
  }
}
```

```
exports.modules = {
  autoRequire: {
    'bundle.js': ['index.js']
  }
}
```

```
exports.paths = {
  public: 'dist'
}
```

.babelrc

```
{
  "presets": ["es2015"]
}
```

Broccoli

Brocfile.js

```
const babel = require(
  'broccoli-babel-transpiler')
const concat = require(
  'broccoli-concat')
```

```
const app = babel('app')

module.exports = concat(app, {
  inputFiles: ['**/*.js'],
  outputFile: '/bundle.js'
})
```

.babelrc

```
{
  "presets": ["es2015"]
}
```

Application

Brunch

index.js

```
const banner = require('banner')

banner.hello()
```

banner.js

```
const name = 'Devfest Lille'
const year = 2017

exports.hello = () => {
  console.log(`Bonjour ${name}.`)
}
```

Broccoli

index.js

hello()

banner.js

```
const name = 'Devfest Lille'
const year = 2017

function hello () {
  console.log(`Bonjour ${name}.`)
}
```

Compilation

Brunch

Le bundle est la transpilation de tous vos fichiers avec une surcouche CommonJS.

Broccoli

Le bundle est la concaténation de tous vos fichiers sans isolation.

Évaluation

Les asset builders sont faciles à mettre en place.

Les asset builders sont faciles à utiliser.

Les asset builders sont faciles à maintenir.

Les asset builders ratent le support de la modularité.

Recommandation

Les asset builders sont adaptés pour les multi-page applications.

Les module bundlers

Les module bundlers sont des outils compatibles avec Node.js, ECMAScript 6, ou TypeScript.

Installation

Webpack

```
npm install -D \
  babel-core \
  babel-loader \
  babel-preset-es2015 \
  webpack
```

Rollup

```
npm install -D \
  babel-preset-es2015 \
  rollup \
  rollup-plugin-babel
```

Exécution

Webpack

webpack

Rollup

rollup -c

Configuration

Webpack

webpack.config.js

```
exports.entry = './app'
```

```
exports.module = {
  rules: [
    {
      test: /\.js$/,
      use: {loader: 'babel-loader'}
    }
  ]
}
```

```
exports.output = {
  filename: 'dist/bundle.js'
}
```

.babelrc

```
{
  "presets": ["es2015"]
}
```

Rollup

rollup.config.js

```
import babel from 'rollup-plugin-babel'
```

```
export default {
  dest: 'dist/bundle.js',
  entry: 'app/index.js',
  format: 'iife',
  plugins: [babel()]
}
```

.babelrc

```
{
  "presets": [
    [
      "es2015",
      {"modules": false}
    ]
  ]
}
```

Application

index.js

```
import {hello} from './banner'

hello()
```

banner.js

```
const name = 'Devfest Lille'
const year = 2017

export function hello () {
  console.log(`Bonjour ${name}.`)
}
```

Compilation

Le bundle est la transpilation de tous vos fichiers en partant d’un point d’entrée.

Évaluation

Les module bundlers sont compliqués à mettre en place.

Les modules bundlers sont compliqués à utiliser.

Les module bundlers sont faciles à maintenir.

Les module bundlers supportent la modularité.

Recommandation

Les modules bundlers sont adaptés pour les single-page applications.

Conclusion

JavaScript n’est pas modulaire.

Il le sera peut-être un jour.

En attendant, on doit coder.

On a besoin de modularité.

Utilisons des bundlers.

Merci de votre attention.