Webpack

Présentation

Selon la documentation de webpack :

> Au coeur de webpack se trouve un "empaqueteur" (bundler) pour les applications javascript modernes. Lorsque webpack traite votre application, il crée en interne un graphique de dépendances qui cartographie chaque module dont votre projet a besoin et génère un ou plusieurs "bundle".

Pour résumer, webpack vous aide à gérer les imports et exports de fichiers de votre projet et il va regrouper tout votre code dans un seul fichier appelé "bundle".

Pour comprendre l'intérêt de webpack, il faut avoir conscience qu'à chaque utilisation d'un fichier dans une page html (css, js, images...) une requête http est effectuée. webpack vous permet donc de minimiser le nombre de requêtes.

Principales fonctionnalités

Possibilité d'utilisation de

- Babel pour transpiler le JS et le rendre ainsi rétrocompatible
- compresseur/minificateur de code en vue de réduire la latence réseau
- compilateur de SASS
- optimisateur d'images
- ...

Premier exemple d'utilisation

Éléments de départ

Imaginons que vous ayez 4 fichiers js

src/first.js

```
export const first = 1
```

src/second.js

```
export const second = 2
```

src/third.js

```
import { first } from './first'
import { second } from './second'

export const third = first + second
```

src/main.js

```
import { third } from './third'

console.log(third)
```

Nous comprenons qu'il va falloir faire attention à l'ordre d'importation des fichiers pour que main.js fonctionne correctement.

C'est justement le travail de webpack !

Initialisation du projet

Créez un nouveau dossier pour ce projet. Vous pouvez l'appeler comme vous le souhaitez.
Ici nous l'appelerons `demo-webpack`.

Dans le dossier `demo-webpack`, créez le dossier `src` et les 4 fichiers javascript présentés précédement.

Enfin, initialiser le projet npm dans le dossier `demo-webpack`:

```
npm init -y
```

A cette étape vous devriez avoir l'arborescence suivante :

```
demo-webpack/
    |_  src/
    |    |_  first.js
```

```
|    |_  second.js
|    |_  third.js
|    |_  main.js
|_  package.json
```

Installation de webpack-cli

```
npm install webpack webpack-cli --save-dev
```

Ensuite, il faut retirer du fichier `package.json` la ligne commençant par `"main"` :

```
{
  "name": "test-cours",
  "version": "1.0.0",
  -"main": "index.js",- // Ligne à retirer
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "devDependencies": {
    "webpack": "^5.94.0",
    "webpack-cli": "^5.1.4"
  }
}
```

On peut dès à présent tester webpack avec la commande suivante :

```
npx webpack ./src/main.js
```

Une fois la commande exécuter on remarque que webpack a généré un nouveau dossier `dist` à la racine du projet et dans ce dossier, un fichier `main.js`.
Ce fichier `main.js` est le bundle généré par webpack. Il contient tout le code de nos 4 fichier javscript source.
Vous pouvez également remarquer que le code a été minifié. C'est à dire que Webpack l'a rassemblé sur une seule ligne et simplifié le code autant que possible afin de réduire la taille du fichier au maximum et ainsi réduire le temps de téléchargement par le navigateur.

Création du fichier de configuration

Afin d'éviter d'avoir à réécrire le chemin de notre fichier javascript principal et pouvoir configurer des outils supplémentaires comme sass, nous allons avoir besoin de créer un fichier de configuration.

Pour ce faire, il vous suffit de créer un nouveau fichier à la racine du projet appelé `webpack.config.js`.

Et voici la configuration à lui donner :

webpack.config.js

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

module.exports = {
  entry: './src/main.js',   // Le fichier js principale de notre code
source.
  output: {
    filename: 'main.js',    // Le nom du bundle à générer
    path: path.resolve(__dirname, 'dist'),   // Le chemin dans lequel le
bundle doit être généré
  },
};
```

Une fois le fichier créé, nous pouvons maintenant simplement appeler la commande suivante :

```
npx webpack
```

Comme vous pouvez le voir, il n'est plus nécessaire de préciser le chemin de notre fichier javascript principal.

Vérification

Afin de vérifier que le bundle fonctionne correctement, vous pouvez l'exécuter avec la commande suivante :

```
node dist/main.js
```

Normalement vous devez obtenir 3.

Loaders

Par défaut, Webpack ne comprend que le JS et le JSON. Les loaders permettent de charger tous les autres types de fichiers afin que Webpack les ajoute à l'arbre des dépendances. Ici nous ne verrons que le loader de sass.

sass-loader

Installation

```
npm install sass-loader css-loader style-loader sass --save-dev
```

Configuration

Modifier le fichier webpack.config.js

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

module.exports = {
  entry: './src/main.js',   // Le fichier js principale de notre code
source.
  output: {
    filename: 'main.js',    // Le nom du bundle à générer
    path: path.resolve(__dirname, 'dist'),   // Le chemin dans lequel le
bundle doit être généré
  },
  module: {
    rules: [
        {
        test: /.scss$/,
        use: ["style-loader", "css-loader", "sass-loader"],
        },
    ],
  }
};
```

Dans le dossier `src`, vous pouvez maintenant créer un dossier `scss` et un fichier `scss/main.scss`

Ensuite, afin que votre style soit intégré dans le bundle, vous pouvez importer le fichier `main.scss` dans le fichier `src/main.js` en ajoutant la ligne suivante en haut du fichier.

```
import './scss/main.scss';
```

Et voila !

Le loader SASS est configuré. A partir de maintenant si votre fichier main.scss contient de code de style valide vous devriez retrouver du code qui ressemble à du code css dans votre le bundle `dist/main.js` après avoir lancer la commande `npx webpack`.

Plugins

Les plugins permettent de réaliser des opérations globales lors de "l'empaquetage" (bundling)

Par exemple HtmlWebpackPlugin simplifie la création de fichier HTML pour servir les bundles webpack.

HtmlWebpackPlugin

Installation

```
npm install html-webpack-plugin --save-dev
```

Création de `src/index.html`

Créez un fihier `src/index.html` :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=, initial-scale=1.0">
    <title>Demo Webpack</title>
</head>
<body>
    <h1>Demo Webpack</h1>
</body>
</html>
```

Modification de webpack.config.js :

```
const HtmlWebpackPlugin = require('html-webpack-plugin');
const path = require('path');

module.exports = {
  entry: './src/main.js',   // Le fichier js principale de notre code
source.
  output: {
    filename: 'main.js',    // Le nom du bundle à généré
    path: path.resolve(__dirname, 'dist'),   // Le chemin dans lequel le
bundle doit être généré
  },
  module: {
    rules: [
        {
        test: /.scss$/,
        use: ["style-loader", "css-loader", "sass-loader"],
        },
    ],
  },
  plugins: [new HtmlWebpackPlugin({
    template: 'src/index.html'
  })],
};
```

Vous verrez qu'au build (`npx webpack`), le fichier dist/index.html est créé.

Si vous ouvrez ce fichier `dist/index.html`, en plus du code qui vient de `src/index.html`, vous pourrez remarquer que le bundle (le fichier `main.js`) a été automatiquement importé.

webpack-dev-server

Le `webpack-dev-server` est un outil qui va nous aider lors du développement :

1. Il permet de lancer un serveur afin de tester notre projet.
2. Il possède une technologie lui permettant de relancer un build dès qu'on modifie notre code source et d'envoyer un signal au navigateur pour qu'il réactualise la page dès que le build est terminé.

Installation

```
npm install webpack-dev-server --save-dev
```

Modification de webpack.config.js

```
const HtmlWebpackPlugin = require('html-webpack-plugin');
const path = require('path');

module.exports = {
  entry: './src/main.js',   // Le fichier js principal de notre code
source.
  mode: 'development',
  output: {
    filename: 'main.js',    // Le nom du bundle à générer
    path: path.resolve(__dirname, 'dist'),   // Le chemin dans lequel le
bundle doit être généré
  },
  module: {
    rules: [
        {
        test: /.scss$/,
        use: ["style-loader", "css-loader", "sass-loader"],
        },
    ],
  },
  plugins: [new HtmlWebpackPlugin({
    template: 'src/index.html'
  })],
};
```

Modification de package.json

```
{
  "name": "test-cours",
  "version": "1.0.0",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "webpack-dev-server",   // Script à ajouter
    "build": "NODE_ENV=production webpack" // Script à ajouter
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "devDependencies": {
    "css-loader": "^7.1.2",
    "html-webpack-plugin": "^5.6.0",
    "sass": "^1.79.3",
    "sass-loader": "^16.0.2",
    "style-loader": "^4.0.0",
    "webpack": "^5.94.0",
    "webpack-cli": "^5.1.4"
  }
}
```

Lancement du serveur de développement

```
npm start
```

Cette commande lance le serveur de développement sur le port 8080: http://localhost:8080/

Vous pouvez ajouter `devServer: {open: true},` dans le fichier webpack.config.js pour que le navigateur s'ouvre automatiquement :

```
const HtmlWebpackPlugin = require('html-webpack-plugin');
const path = require('path');

module.exports = {
    entry: './src/main.js',   // Le fichier js principal de notre code
source.
    mode: 'development',
    output: {
        filename: 'main.js',    // Le nom du bundle à générer
        path: path.resolve(__dirname, 'dist'),   // Le chemin dans lequel
le bundle doit être généré
    },
    module: {
        rules: [
            {
                test: /.scss$/,
```

```
                use: ["style-loader", "css-loader", "sass-loader"],
            },
        ],
    },
    plugins: [new HtmlWebpackPlugin({
        template: 'src/index.html',
    })],
    devServer: {open: true},
};
```

Conclusion

Webpack est un outil puissant pour les développeurs travaillant avec des applications JavaScript modernes. Il permet de centraliser et d'optimiser le code source en un seul bundle, réduisant ainsi le nombre de requêtes HTTP nécessaires et améliorant les performances globales de l'application. Grâce à ses fonctionnalités telles que les loaders, les plugins et l'intégration d'un serveur de développement, Webpack offre une flexibilité et une efficacité inégalées pour la gestion de projets complexes. Il facilite également l'utilisation de technologies comme SASS, la transpilation JavaScript avec Babel, et l'optimisation d'images, ce qui en fait un choix incontournable pour les projets front-end modernes.