4 minutes

Utilisation de polyfills avec Webpack

Les `polyfills`

Un `polyfill` est du code d'une version plus ancienne de `JavaScript` qui permet de simuler une fonctionnalité récente du langage et qui n'est pas encore disponible dans toutes les versions des navigateurs.

Par exemple la méthode `Promise.allSettled()` qui est standardisée depuis `ECMAScript2020` et qui n'est pas encore implémentée sur `Edge` peut avoir comme `polyfill` :

```
Promise.allSettled = function(promises) {
  return Promise.all(promises.map(p => Promise.resolve(p).then(value => ({
    state: 'fulfilled',
    value
  }), reason => ({
    state: 'rejected',
    reason
  }))));
};
```

Configuration de `Webpack` pour utiliser des polyfills

Heureusement nous n'avons pas à ajouter nous-même tous les `polyfills` nécessaires !

`Webpack` et `Babel` vont le faire pour nous. Mais ils ont besoin d'une librairie `core-js`

Cette librairie est une immense liste de `polyfills`.

Nous avons également besoin de `regenerator-runtime` qui est une librairie maintenue par `Facebook` et qui permet notamment de permettre le `polyfill` en `ES5` des fonctions asynchrones que nous allons utiliser.

Vous devez donc installer les deux librairies :

```
npm i -D core-js@3 regenerator-runtime
```

Modification du fichier de configuration `babel`

Maintenant que nous avons les librairies nécessaires, nous allons modifier le fichier `babel.config.js` :

```
const presets = [
  [
    "@babel/preset-env",
    {
      useBuiltIns: "usage",
      debug: true,
      corejs: 3,
      targets: "> 0.25%, not dead"
    }
  ]
];

module.exports = { presets };
```

Comme vous pouvez le voir, nous passons en plus un objet d'options que nous allons détailler.

L'option **`useBuiltIns`** permet de gérer l'utilisation des `polyfills` par `Babel`. Par défaut l'option est à `false` et `Babel` ne va pas ajouter de `polyfills` automatiquement.

La valeur `usage` permet d'importer les `polyfills` nécessaires pour les versions ciblées des navigateurs pour les fonctionnalités utilisées (à la différence d'`entry` qui importera tous les `polyfills` nécessaires pour les versions des navigateurs ciblées sans distinction de l'utilisation).

L'option `corejs` permet simplement de spécifier quelle version de la librairie `core-js` nous utilisons.

L'option `debug` permet d'indiquer quels sont les versions des navigateurs ciblés par votre requête et tous les `polyfills` nécessaires pour votre code.

L'option `targets` est une requête permettant de définir les navigateurs ciblés.

Elle utilise en fait la librairie `Browserslist` qui est une dépendance de `Babel`.

Elle est extrêmement puissante car vous pouvez quasiment tout faire.

Vous souhaitez supporter les navigateurs utilisez en France par au moins 0.2% des gens ?

Aucun problème : `"> 0.2% in FR"`.

Vous souhaitez supporter toutes les versions des navigateurs représentant au moins 1% des utilisateurs ou qui ne sont pas morts ?

Utilisez : `"> 1%, not dead"`.

Dernières trois versions des navigateurs ?

Utilisez : `"last 3 versions"`.

Utiliser la requête recommandée ?

Utilisez : `"defaults"` qui équivaut à `"> 0.5%, last 2 versions, Firefox ESR, not dead"`.

Les navigateurs `dead` sont ceux qui ne sont plus supportés : aucune mise à jour depuis plus de 24 mois et qui sont définitivement abandonnés (par exemple `IE 10`, `BlackBerry 10`).

Ces requêtes se traduisent en définitives par une liste de navigateurs avec une version minimale à supporter :


```
{
  "android": "76",
  "chrome": "49",
  "edge": "17",
  "firefox": "68",
  "ie": "11",
  "ios": "12.2",
  "opera": "46",
  "safari": "5.1",
  "samsung": "9.2"
}
```


Cela signifie ici que toutes les versions de `Chrome` depuis la 49 seront supportées.

Avec l'option `debug` à `true` vous pouvez voir le résultat de votre requête en terme de versions de navigateur minimales supportées.

Ensuite, vous aurez la liste de tous les `polyfills` nécessaires pour votre application :


```
Using plugins:
  transform-template-literals { "android":"76", "ie":"11", "ios":"12.2",
"safari":"5.1" }
  transform-literals { "android":"76", "ie":"11", "safari":"5.1" }
  transform-function-name { "android":"76", "chrome":"49", "edge":"17", "ie":"11",
"safari":"5.1" }
  transform-arrow-functions { "android":"76", "ie":"11", "safari":"5.1" }
  transform-block-scoped-functions { "android":"76", "safari":"5.1" }
  transform-classes { "android":"76", "ie":"11", "safari":"5.1" }
  transform-object-super { "android":"76", "ie":"11", "safari":"5.1" }
  transform-shorthand-properties { "android":"76", "ie":"11", "safari":"5.1" }
  transform-duplicate-keys { "android":"76", "ie":"11", "safari":"5.1" }
  transform-computed-properties { "android":"76", "ie":"11", "safari":"5.1" }
```

```
transform-for-of { "android":"76", "chrome":"49", "ie":"11", "safari":"5.1" }
transform-sticky-regex { "android":"76", "ie":"11", "safari":"5.1" }
```

A chaque fois, vous avez le nom du `polyfill` puis la liste des navigateurs nécessitant ce `polyfill`.

Plus il y a de `polyfills` plus votre application sera lourde mais plus elle sera supportée par tous les navigateurs.

Il n'y a pas de recommandation, tout dépend de votre cible !

Si vous n'avez pas de cible particulière utilisez `defaults` qui est la recommandation officielle de `Browserslist`.