ECMAScript 2015

Les modules

Jusqu’à maintenant

Nativement

- Nous l'avons vu, JavaScript ne possédait que 2 scopes :
  - Global
  - Fonction

- Si une variable ou une fonction est définie en dehors d'une fonction, elle est donc globale

➔ Source de bugs en écrasant les valeurs dans un autre fichier

Solution 1

- Les namespaces :

```
window.myNamespace = {};

myNamespace.myVar = "test";
myNamespace.myFunction = function() {};
```

➔ Très verbeux et nécessite beaucoup de rigueurs

Solution 2

- Les fonctions auto executées :

```
(function() {
  var myVar = "test";
  var myFunction = function() {};
})();
```

➔Beaucoup mieux mais encore on peut faire mieux

Solution 3

- Les modules !
- Il existe deux « pseudo-standards » en concurrence :
  - AMD (Asynchronous Module Definition)
  - CommonJs
- Les deux approches nécessitent des librairies pour fonctionner
- CommonJs est utilisé par NodeJs
- Les déclarations sont plutôt verbeuses puisqu'ils ne peuvent pas profiter de la syntaxe native
- Tout ce qui est déclaré dans un module est isolé du reste
- Il est possible d'importer des modules depuis un module

Solution 3a

- AMD :

```
define(['module2'] , function (myImportedVar) {
  var internalVar = "internal";
  var externalVar = "external";

  return externalVar;
});
```

➔ Nécessite requireJs

Solution 3b

- CommonJs :

```
var myImportedVar = require('module2');

var internalVar = "internal";
var externalVar = "external";

module.exports = externalVar;
```

➔ Fonctionne dans nodeJs ou avec browserifiy dans le navigateur

Solution 3c

- UMD :

```
(function (root, factory) {
  if (typeof define === 'function' && define.amd) {
      define(['module2'], factory);
  } else if (typeof module === 'object' &&
module.exports) {
       module.exports = factory(require('module2'));
  } else {
    root.returnExports =
factory(root.myImportedVar);
  }
}(this, function (myImportedVar) {
  return {};
}));
```

➔ Fonctionne en AMD, commonJS et en variables globales…

Modules natifs

Nouvelle syntaxe : export

- Il est possible d'exporter n'importe quel données avec le mot clef **export** :

```
export var test = "test";
export const test = "test";
export function test() { return "test" }
export var test = /test/;
export class test {}
```

- Vous pouvez mettre autant d'export que vous le souhaitez dans un même module

Nouvelle syntaxe : import

- Il est possible d'importer un module avec **import** :

```
import {test} from 'module';
import {test,otherVar} from 'module';
import * as namespace from 'module';
import {test as anotherName} from 'module';
```

- Les noms des imports doivent correspondre aux noms exportés
- On renomme avec 'as'

Export par défaut

- Chaque module peut contenir UN seul module par défaut :

```
export default "test";
export default function() {};
```

- Vous pourrez l'importer sans les accolades :

```
import test from 'module';
```

Ce qu'il faut savoir

- Un module est un fichier javascript
- Seul ce qui est exporté est visible en dehors du module
- Les modules sont des singletons, a chaque import vous récupérez la même instance
- Les imports doivent forcément se trouver au début du fichier
- Les exports peuvent se trouver n'importe où
- Les modules importés sont des « vues »
- Par conséquent les dépendances cycliques ne posent pas de problèmes
- Les imports sont statiques
- Une norme pour des imports dynamiques est en cours de spécification

Module en tant « vue »

```
//------ lib.js ------
export let counter = 3;
export function incCounter() {
  counter++;
}

//------ main1.js ------
import { counter, incCounter } from './lib';

// The imported value `counter` is live
console.log(counter); // 3
incCounter();
console.log(counter); // 4

// The imported value can't be changed
counter++; // TypeError
```

Dans le navigateur

- Nouveau type de script :

```
<script type="module" src="module.js"></script>
```

Le problème

- Les modules ne sont supportés (pour le moment) nulle part :
  - Aucun navigateur ne les supportent nativement
  - NodeJs ne les supporte pas nativement

➔ Babel les compile par défaut en CommonJs (ou en AMD via configuration) pour les rendre compatible avec le reste

L’outil ultime ? Webpack

Qu’est ce que webpack ?

- Gère les modules dans tous les formats et les fait cohabités
- Package votre site en un ou plusieurs fichiers agrégés
- Utilise des plugins pour appliquer des transformations (par exemple compiler vos modules avec babel)
- Offre un serveur web avec live-reload

Alternative : SystemJs

Installation

Pour pouvoir obtenir la ligne de commande :

npm install –g webpack

Pour pouvoir l'utiliser dans son projet (en + du global)

npm install --save-dev webpack

Puis créer un fichier webpack.config.js à la racine du projet

Fichier webpack.config.js

```
var webpack = require('webpack');

module.exports = {
   entry: "./site/main.js",
   output: {
      path: __dirname,
      filename: "./site/bundle.js"
   },
   devtool: "source-map",
   module: {
      loaders: []
   }
};
```

Ajouter babel

- Installer le babel-loader

npm install --save-dev babel-loader

Un loader permet de transformer des fichiers lors du build

Modification du fichier webpack

```
var webpack = require('webpack');

module.exports = {
   entry: "./site/main.js",
   output: {
      path: __dirname,
      filename: "./site/bundle.js"
   },
   devtool: "source-map",
   module: {
      loaders: [{
         test: /\.js$$/, loader: "babel-loader"
      }]
   }
};
```

Configurer babel directement dans webpack

- Notre projet utilise le fichier .babelrc, mais il est possible de configurer babel directement dans webpack :

```
loaders: [{
   test: /\.js$$/,
   loader: "babel-loader",
   query: {
     presets: [ "es2015"],
     cacheDirectory:true
   }
}]
```

Utilisation de webpack-dev-server

- Pour utiliser webpack en développement, le webpac-dev-server est recommandé
- Il permet de :
  - Lancer un serveur web avec les fichiers buildés
  - Recompiler le code qui est modifier automatiquement
  - Dans certains cas faire du rechargement à chaud ou du live reload

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

webpack-dev-server --hot --inline
```