eBook Gratuit

APPRENEZ Web Component

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#web-component

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: web-component

It is an unofficial and free Web Component ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Web Component.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec le composant Web

Remarques

Cette section fournit une vue d'ensemble de ce que sont les composants Web et pourquoi un développeur peut vouloir les utiliser.

Les composants Web sont un ensemble de nouvelles technologies Web implémentées dans les navigateurs Web modernes et utilisés pour concevoir des éléments Web réutilisables avec la seule aide de HTML, JavaScript et CSS.

Les sujets couverts par le terme *Web Compoments* sont les suivants:

- Éléments personnalisés
- Modèles HTML
- DOM ombre
- Importations HTML

Ces technologies sont complémentaires et peuvent être utilisées ensemble ou séparément.

Versions

Composants	spécification	Dernière sortie
Modèles HTML	Recommandation HTML5 du W3C	2014-10-28
Éléments personnalisés	W3C Working Drafts ou WHATWG HTML et DOM Living Standard	2016-10-13
DOM ombre	W3C Working Drafts ou WHATWG HTML et DOM Living Standard	2017-01-16
Importations HTML	Projets de travail du W3C	2016-02-25

Examples

Disponibilité

Implémentations natives

L'élément `<template>` est implémenté dans tous les navigateurs modernes:

- Chrome,
- Bord,
- Firefox,
- Opéra,
- Safari,
- ...

Les éléments personnalisés `customElements.define()` , Shadow DOM `attachShadow()` et les importations HTML `<link rel="import">` sont implémentés dans les dernières versions de Chrome et Opera.

Polyfills

Pour les autres navigateurs, vous pouvez utiliser une bibliothèque polyfill:

- pour les éléments personnalisés: à partir de WebReflection ou Webcomponents.org ,
- pour Shadow DOM: à partir de Webcomponents.org ,
- pour le modèle: de Neovov ,
- pour les importations HTML: à partir de Webcomponents.org

Modèle HTML - Bonjour tout le monde

Utilisez un élément `<template>` pour concevoir un modèle HTML que vous pouvez ensuite réutiliser dans votre code.

```
<template id="Template1">
    Hello, World !
<template>

<div id="Target1"></div>

<script>
   Target1.appendChild( Template1.content.cloneNode( true ) )
</script>
```

Cela va insérer le contenu du modèle dans le div `#Target1` .

Élément personnalisé - Hello World

Créez une nouvelle balise HTML nommée `<hello-world>` qui affichera "Hello, World!":

```
<script>
//define a class extending HTMLElement
class HelloWorld extends HTMLElement {
    connectedCallback () {
      this.innerHTML = 'Hello, World!'
    }
}

//register the new custom element
customElements.define( 'hello-world', HelloWorld )
</script>
```

```
<!-- make use the custom element -->
<hello-world></hello-world>
```

Shadow DOM - Hello World

Ajouter un DOM Shadow à un `div` qui affichera "Hello, World!" au lieu de son contenu initial.

```
<div id="Div1">intial content</div>

<script>
   var shadow = Div1.attachShadow( { mode: 'open' } )
   shadow.innerHTML = "Hello, World!"
</script>
```

Importation HTML - Bonjour tout le monde

Importez un fichier HTML qui ajoutera un div avec "Hello, World!" à la fin de l'arborescence DOM du document principal.

Fichier importé *hello.html* :

```
<script>
   var div = document.createElement( 'div' )
   div.innerHTML = 'Hello, World!'
   document.body.appendChild( div )
</script>
```

Fichier principal *index.html* :

```
<html>
  <link rel="import" href="hello.html">
```

Bonjour exemple mondial

Cet exemple combine un élément personnalisé, un modèle, un DOM Shadow et une importation HTML pour afficher un "Hello, World!" chaîne en HTML.

Dans le fichier `hello-world.html` :

```
<!-- 1. Define the template -->
<template>
   Hello, World!
</template>

<script>
  var template = document.currentScript.ownerDocument.querySelector( 'template' )

  //2. Define the custom element

  customElements.define( 'hello-world', class extends HTMLElement
  {
```

```
      constructor()
      {
          //3. Create a Shadow DOM
          var sh = this.attachShadow( { mode: 'open' } )
          sh.appendChild( document.importNode( template.content, true ) )
      }
  } )
</script>
```

Dans le fichier principal `index.html` :

```
<html>
<head>
    <!-- 4. Import the HTML component -->
    <link rel="import" href="hello-world.html">
</head>
<body>
    <hello-world></hello-world>
</body>
</html>
```

Lire Démarrer avec le composant Web en ligne: https://riptutorial.com/fr/web-component/topic/8239/demarrer-avec-le-composant-web

Chapitre 2: Test de composants Web

Introduction

Choses à considérer lorsque nous voulons tester nos composants avec: Styles, Modèles, Classes de composants.

Examples

Webpack et Jest

Jest est utilisé par Facebook pour tester tous les codes JavaScript, y compris les applications React. L'une des philosophies de Jest est de fournir une expérience intégrée «sans configuration». Nous avons observé que lorsque les ingénieurs disposent d'outils prêts à l'emploi, ils finissent par écrire davantage de tests, ce qui permet d'obtenir des bases de code plus stables et plus saines.

Un exemple de travail complet est disponible sur GitHub en tant que web-components-webpack-es6-boilerplate

Jest exécute des tests dans l'environnement NodeJS avec jsdom . L'ensemble du processus est facile. Considérons la configuration du webpack , en supposant que notre structure de projet ressemble à un exemple suivant:

```
-src
    --client
    --server
-webpack
    --config.js
package.json
```

Une structure de répertoire simple conçue pour séparer la logique de `server render` du `server render` du reste. **Le** fichier **Webpack** `config.js` contiendrait les modules suivants:

```
  resolve: {
    modules: ["node_modules"],
    alias: {
      client: path.join(__dirname, "../src/client"),
      server: path.join(__dirname, "../src/server")
    },
    extensions: [".js", ".json", ".scss"]
  },
```

Nous pouvons configurer **Jest** pour refléter notre configuration **Webpack** .

```
module.exports = {
  setupTestFrameworkScriptFile: "<rootDir>/bin/jest.js",
  mapCoverage: true,
```

```
  moduleFileExtensions: ["js", "scss", "html"],
  moduleDirectories: ["node_modules"],
  moduleNameMapper: {
    "src/(.*)$": "<rootDir>/src/$1"
  },
  transform: {
    "^.+\\.(js|html|scss)$": "<rootDir>/bin/preprocessor.js"
  },
  testMatch: ["<rootDir>/test/**/?(*.)(spec|test).js"],
  testPathIgnorePatterns: ["<rootDir>/(node_modules|bin|build)"]
};
```

Où devrions-nous enregistrer cette configuration?

Nous pouvons le faire dans le fichier `package.json` sous la clé `jest` ou créer comme dans cet exemple le fichier `jest.config.js` dans la racine du projet.

Ce que nous voulons réaliser, c'est de nous assurer que nos fichiers `html` seront importés correctement. Cela signifie en leur échappant avec un `preprocessor` personnalisé, car utiliser uniquement `babel-jest` génère une erreur lors de la tentative d'analyse des fichiers non `js` .

L'autre chose importante ici est le script `setupTestFrameworkScriptFile` qui inclut en réalité `custom elements` polyfills d' `custom elements` à `jsdom` . Voici à quoi ressemble notre `preprocessor.js` :

```
const babelJest = require("babel-jest");

const STYLE_URLS_REGEX = /styles:\s*\[\s*((?:'|").*\s*(?:'|")).*\s*.*\]/g;
const ESCAPE_TEMPLATE_REGEX = /(\${|\`)/g;

module.exports.process = (src, path, config) => {
  if (path.endsWith(".html")) {
    src = src.replace(ESCAPE_TEMPLATE_REGEX, "\\$1");
    src = "module.exports=`" + src + "`;";
  }
  src = src.replace(STYLE_URLS_REGEX, "styles: []");

  return babelJest.process(src, path, config);
};
```

Ce que fait ce script est simple: supprimer le contenu des fichiers de style car nous n'avons pas besoin de le tester et des modèles d'échappement, par exemple lorsque nous les importons avec la syntaxe de `require('template.html')` . Ensuite, il transmet le contenu à babel transformateur.

La dernière chose importante à faire est d'inclure `web components polyfills web components` . Par défaut, `jsdom` ne les supporte pas encore. Pour ce faire, nous pouvons simplement ajouter `setupTestFrameworkScriptFile` dans notre exemple, c'est `jest.js` avec le contenu suivant:

```
require("document-register-element/pony")(window);
```

De cette façon, nous pouvons accéder à `web components` **API des** `web components` dans `jsdom` .

Après avoir tout configuré, nous devrions avoir une structure comme celle-ci:

```
-bin
    --jest.js
    --preprocessor.js
-src
    --client
    --server
-webpack
    --config.js
-test
package.json
jest.config.js
```

Où nous gardons nos tests dans le répertoire de `test` et pouvons l'exécuter avec la commande:

```
yarn run jest --no-cache --config $(node jest.config.js) .
```

Lire Test de composants Web en ligne: https://riptutorial.com/fr/web-component/topic/10057/test-de-composants-web

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec le composant Web	Community, Mike, Supersharp
2	Test de composants Web	Vardius