EBook Gratis

APRENDIZAJE Web Component

Free unaffiliated eBook created from
Stack Overflow contributors.

#web-component

Tabla de contenido

Acerca de

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

Capítulo 1: Comenzando con el componente web

Observaciones

Esta sección proporciona una descripción general de qué son los componentes web y por qué un desarrollador puede querer usarlos.

Los componentes web son un conjunto de nuevas tecnologías web implementadas en navegadores web modernos y se utilizan para diseñar elementos web reutilizables con la única ayuda de HTML, JavaScript y CSS.

Los temas cubiertos por el término *Web Compoments* son:

- Elementos personalizados
- Plantillas HTML
- Shadow DOM
- Importaciones HTML

Estas tecnologías son complementarias y se pueden utilizar juntas o por separado.

Versiones

Componentes	Especificación	Último lanzamiento
Plantillas HTML	Recomendación HTML5 del W3C	2014-10-28
Elementos personalizados	Borradores de trabajo del W3C o WHATWG HTML y DOM Living Standard	2016-10-13
Shadow DOM	Borradores de trabajo del W3C o WHATWG HTML y DOM Living Standard	2017-01-16
Importaciones HTML	Borradores de trabajo del W3C	2016-02-25

Examples

Disponibilidad

Implementaciones nativas

El elemento `<template>` se implementa en todos los navegadores modernos:

- Cromo,
- Borde,
- Firefox,
- Ópera,
- Safari,
- ...

Elementos personalizados `customElements.define()` , Shadow DOM `attachShadow()` y HTML Imports `<link rel="import">` se implementan en las últimas versiones de Chrome y Opera.

Polyfills

Para otros navegadores, puede utilizar una biblioteca de polyfill:

- para elementos personalizados: de WebReflection o Webcomponents.org ,
- para Shadow DOM: de Webcomponents.org ,
- Para la plantilla: de Neovov ,
- para Importaciones HTML: de Webcomponents.org

Plantilla HTML - Hola Mundo

Use un elemento `<template>` para diseñar una plantilla HTML que luego pueda reutilizar en su código.

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

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

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

Esto insertará el contenido de la plantilla en la `#Target1` .

Elemento personalizado - Hello World

Cree una nueva etiqueta HTML llamada `<hello-world>` que mostrará "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 - Hola Mundo

Agrega un DOM Shadow a un `div` que mostrará "Hello, World!" En lugar de su contenido inicial.

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

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

Importación HTML - Hola Mundo

Importe un archivo HTML que agregará un div con "Hello, World!" Al final del árbol DOM del documento principal.

Archivo importado *hello.html* :

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

Archivo principal *index.html* :

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

Hola mundo ejemplo

Este ejemplo combina Elemento personalizado, Plantilla, DOM de sombra y Importación de HTML para mostrar el mensaje "¡Hola, mundo!" cadena en HTML.

En el archivo `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>
```

En el archivo 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>
```

Lea Comenzando con el componente web en línea: https://riptutorial.com/es/web-component/topic/8239/comenzando-con-el-componente-web

Capítulo 2: Pruebas de componentes web

Introducción

Cosas a considerar cuando queremos probar nuestros componentes con: Estilos, Plantillas, Clases de componentes.

Examples

Webpack y broma

Jest es utilizado por Facebook para probar todo el código JavaScript, incluidas las aplicaciones React. Una de las filosofías de Jest es proporcionar una experiencia integrada de "configuración cero". Observamos que cuando a los ingenieros se les proporcionan herramientas listas para usar, terminan escribiendo más pruebas, lo que a su vez da como resultado bases de código más estables y saludables.

El ejemplo de trabajo completo está disponible en GitHub como web-components-webpack-es6-boilerplate

Jest ejecuta pruebas en el entorno NodeJS con jsdom . Todo el proceso es fácil. Consideremos la siguiente configuración del paquete web , asumiendo que la estructura de nuestro proyecto se parece al siguiente ejemplo:

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

Una estructura de directorio simple diseñada para separar la lógica de `server render` del `server render` del resto. **El** archivo **webpack** `config.js` contendría los siguientes módulos:

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

Podemos configurar **Jest** para reflejar nuestra configuración de **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)"]
};
```

¿Dónde deberíamos guardar esta configuración?

Podemos hacerlo en el archivo `package.json` bajo la clave `jest` o crear como en este ejemplo el archivo `jest.config.js` en la raíz del proyecto.

Lo que queremos lograr es asegurarnos de que nuestros archivos `html` se importarán correctamente. Eso significa que evitándolos con `preprocessor` personalizado, ya que usar solo `babel-jest` generaría un error al intentar analizar archivos que no sean `js` .

La otra cosa importante aquí es el script `setupTestFrameworkScriptFile` que en realidad incluye `custom elements` polyfills a `jsdom` . Así es como se ve nuestro `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);
};
```

Lo que hace este script es simple: elimine el contenido de los archivos de estilo, ya que no necesitamos / queremos probarlo, y evite las plantillas, cuando las importamos, por ejemplo, con la sintaxis `require('template.html')` . Luego pasa el contenido a babel transformer.

Lo último que hay que hacer es incluir `web components polyfills` . Como por defecto `jsdom` no los soporta todavía. Para hacerlo, simplemente podemos agregar `setupTestFrameworkScriptFile` en nuestro ejemplo, es `jest.js` con el siguiente contenido:

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

De esta manera podemos acceder a `web components` **API de** `web components` en `jsdom` .

Después de configurar todo deberíamos tener una estructura como esta:

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

Donde guardamos nuestras pruebas en el directorio de `test` y podemos ejecutarlo con el comando: `yarn run jest --no-cache --config $(node jest.config.js) .`

Lea Pruebas de componentes web en línea: https://riptutorial.com/es/web-component/topic/10057/pruebas-de-componentes-web

Creditos

S. No	Capítulos	Contributors
1	Comenzando con el componente web	Community, Mike, Supersharp
2	Pruebas de componentes web	Vardius