EBook Gratuito

APPRENDIMENTO Web Component

Free unaffiliated eBook created from
Stack Overflow contributors.

#web-component

Sommario

Di

Capitolo 1: Iniziare con Web Component

Osservazioni

Questa sezione fornisce una panoramica di quali sono i componenti Web e perché uno sviluppatore potrebbe volerli utilizzare.

I componenti Web sono un insieme di nuove tecnologie Web implementate nei moderni browser Web e sono utilizzate per progettare elementi web riutilizzabili con l'unico aiuto di HTML, JavaScript e CSS.

Gli argomenti trattati con il termine *Web Compoments* sono:

- Elementi personalizzati
- Modelli HTML
- DOM ombra
- Importa HTML

Queste tecnologie sono complementari e possono essere utilizzate insieme o separatamente.

Versioni

componenti	specificazione	Ultima versione
Modelli HTML	Raccomandazione W3C HTML5	2014/10/28
Elementi personalizzati	W3C Working Drafts o WHATWG HTML e DOM Living Standard	2016/10/13
DOM ombra	W3C Working Drafts o WHATWG HTML e DOM Living Standard	2017/01/16
Importa HTML	W3C Working Drafts	2016/02/25

Examples

Disponibilità

Implementazioni native

L'elemento `<template>` è implementato in tutti i browser moderni:

- Cromo,
- Bordo,
- Firefox,
- Musica lirica,
- Safari,
- ...

Elementi personalizzati `customElements.define()` , Shadow DOM `attachShadow()` e HTML Imports `<link rel="import">` sono implementati nelle ultime versioni di Chrome e Opera.

polyfills

Per altri browser, puoi utilizzare una libreria di polyfill:

- per elementi personalizzati: da WebReflection o Webcomponents.org ,
- per Shadow DOM: da Webcomponents.org ,
- per modello: da Neovov ,
- per le importazioni HTML: da Webcomponents.org

Modello HTML - Hello World

Utilizza un elemento `<template>` per progettare un modello HTML che puoi riutilizzare nel codice.

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

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

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

Questo inserirà il contenuto del modello nel div `#Target1` .

Elemento personalizzato - Hello World

Crea un nuovo tag HTML denominato `<hello-world>` che visualizzerà "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

Aggiungi un DOM ombra a un `div` che visualizzerà "Hello, World!" invece del suo contenuto iniziale.

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

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

Importazione HTML - Hello World

Importa un file HTML che aggiungerà un div con "Hello, World!" alla fine dell'albero DOM del documento principale.

File importato *hello.html* :

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

File principale *index.html* :

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

Ciao esempio del mondo

Questo esempio combina elementi personalizzati, modelli, DOM ombra e importazione HTML per visualizzare un "Hello, World!" stringa in HTML.

Nel file `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>
```

Nel file principale `index.html` :

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

Leggi Iniziare con Web Component online: https://riptutorial.com/it/web-component/topic/8239/iniziare-con-web-component

Capitolo 2: Test dei componenti Web

introduzione

Cose da considerare quando vogliamo testare i nostri componenti con: Stili, Modelli, Classi di componenti.

Examples

Webpack e Jest

Jest viene utilizzato da Facebook per testare tutto il codice JavaScript, incluse le applicazioni React. Una delle filosofie di Jest è fornire un'esperienza integrata di "zero-configurazione". Abbiamo osservato che quando gli ingegneri dispongono di strumenti pronti all'uso, finiscono per scrivere più test, il che a sua volta porta a basi di codice più stabili e salutari.

L'esempio operativo completo è disponibile su GitHub come web-components-webpack-es6-boilerplate

Jest esegue test in ambiente NodeJS con jsdom . L'intero processo è facile. Consideriamo la seguente configurazione del webpack , assumendo che la nostra struttura di progetto assomigli al seguente esempio:

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

Una semplice struttura di directory progettata per separare la logica di `server render` del `server render` dal resto. `config.js` file `config.js` **Webpack** conterrà i seguenti moduli:

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

Possiamo impostare **Jest** per riflettere la nostra configurazione del **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)"]
};
```

Dove dovremmo salvare questa configurazione?

Possiamo farlo nel file `package.json` sotto la chiave `jest` o creare come in questo esempio il file `jest.config.js` nella root del progetto.

Quello che vogliamo ottenere è assicurarsi che i nostri file `html` vengano importati correttamente. Ciò significa evadendoli con un `preprocessor` personalizzato, poiché l'utilizzo di `babel-jest` genera un errore quando si tenta di analizzare i file non `js` .

L'altra cosa importante qui è `setupTestFrameworkScriptFile` script `setupTestFrameworkScriptFile` che in realtà include `custom elements` polyfill degli `custom elements` su `jsdom` . Ecco come appare il nostro `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);
};
```

Quello che fa questo script è semplice: rimuovi il contenuto dei file di stile in quanto non è necessario / volerlo testare e sfuggire i modelli, quando li importiamo ad esempio con la sintassi `require('template.html')` . Quindi passa il contenuto al trasformatore Babel.

L'ultima cosa importante da fare è includere `web components` `polyfills` `web components` . Come per impostazione predefinita `jsdom` non li supporta ancora. Per farlo, possiamo semplicemente aggiungere `setupTestFrameworkScriptFile` nel nostro esempio è `jest.js` con il seguente contenuto:

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

In questo modo possiamo accedere **all'API dei** `web components` in `jsdom` .

Dopo aver impostato tutto, dovremmo avere una struttura come questa:

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

Dove teniamo i nostri test nella directory di `test` e possiamo eseguirlo con il comando: `yarn run jest --no-cache --config $(node jest.config.js) .`

Leggi Test dei componenti Web online: https://riptutorial.com/it/web-component/topic/10057/test-dei-componenti-web

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con Web Component	Community, Mike, Supersharp
2	Test dei componenti Web	Vardius