Бесплатная электронная книга

УЧУСЬ

Web Component

Free unaffiliated eBook created from
Stack Overflow contributors.

#web-component

Около

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

глава 1: Начало работы с веб-компонентом

замечания

В этом разделе представлен обзор того, что такое веб-компоненты, и почему разработчик может захотеть их использовать.

Веб-компоненты - это набор новых веб-технологий, реализованных в современных веб-браузерах, и используется для создания многоразовых веб-элементов с помощью только HTML, JavaScript и CSS.

Темы, охватываемые термином *Web Compoments* :

- Пользовательские элементы
- Шаблоны HTML
- Тень DOM
- Импорт HTML

Эти технологии являются взаимодополняющими и могут использоваться вместе или по отдельности.

Версии

Компоненты	Спецификация	Последний выпуск
Шаблоны HTML	Рекомендация W3C HTML5	2014-10-28
Пользовательские элементы	W3C Working Drafts или WHATWG HTML и DOM Living Standard	2016-10-13
Тень DOM	W3C Working Drafts или WHATWG HTML и DOM Living Standard	2017-01-16
Импорт HTML	Рабочие проекты W3C	2016-02-25

Examples

Доступность

Нативные реализации

Элемент `<template>` реализуется во всех современных браузерах:

- Хром,
- Край,
- Fire Fox,
- Opera,
- Сафари,
- ...

Пользовательские элементы `customElements.define()` , Shadow DOM `attachShadow()` и HTML Imports `<link rel="import">` реализованы в последних версиях Chrome и Opera.

Polyfills

Для других браузеров вы можете использовать библиотеку polyfill:

- для пользовательских элементов: из WebReflection или Webcomponents.org ,
- для Shadow DOM: от Webcomponents.org ,
- для шаблона: от Неова ,
- для импорта HTML: из Webcomponents.org

Шаблон HTML - Hello World

Используйте элемент `<template>` для создания HTML-шаблона, который затем можно повторно использовать в коде.

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

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

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

Это `#Target1` содержимое шаблона в `#Target1` div.

Пользовательский элемент - Hello World

Создайте новый HTML-тег с именем `<hello-world>` , который отобразит «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>
```

Тень DOM - Hello World

Добавьте Shadow DOM в `div` , который отобразит «Hello, World!». вместо его первоначального контента.

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

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

Импорт HTML - Hello World

Импортируйте HTML-файл, который добавит div с помощью «Hello, World!». в конце дерева DOM основного документа.

Импортированный файл *hello.html* :

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

Основной файл *index.html* :

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

Пример Hello World

В этом примере сочетаются пользовательский элемент, шаблон, теневая DOM и HTML Import, чтобы отобразить «Hello, World!». строка в HTML.

В файле `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>
```

В основном файле `index.html` :

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

Прочитайте Начало работы с веб-компонентом онлайн: https://riptutorial.com/ru/web-component/topic/8239/начало-работы-с-веб-компонентом

глава 2: Тестирование веб-компонентов

Вступление

Что нужно учитывать, когда мы хотим протестировать наши компоненты с помощью: стилей, шаблонов, классов компонентов.

Examples

Webpack и Jest

Jest используется Facebook для тестирования всего кода JavaScript, включая приложения React. Одна из философий Джест состоит в том, чтобы обеспечить интегрированный опыт «нулевой конфигурации». Мы заметили, что, когда инженеры снабжены готовыми инструментами, они в конечном итоге пишут больше тестов, что, в свою очередь, приводит к более стабильным и здоровым кодам.

Полный рабочий пример доступен на GitHub как web-компоненты-webpack-es6-шаблон

Jest запускает тесты в среде NodeJS с помощью jsdom . Весь процесс прост. Рассмотрим следующую настройку webpack , предполагая, что наша структура проекта выглядит следующим образом:

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

Простая структура каталогов, предназначенная для отделения `server render` от логики вывода от остальных. Файл `config.js` **Webpack** будет содержать следующие модули:

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

Мы можем настроить **Jest,** чтобы отобразить нашу конфигурацию **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)"]
};
```

Где мы должны сохранить эту конфигурацию?

Мы можем сделать это в файле `package.json` под ключом `jest` или создать как в этом примере файл `jest.config.js` в корне проекта.

Мы хотим добиться того, чтобы наши `html` файлы были импортированы правильно. Это означает, что вы избегаете их с помощью пользовательского `preprocessor` , поскольку использование только `babel-jest` приведет к ошибке при попытке проанализировать файлы не `js` .

Другое важное значение здесь - `setupTestFrameworkScriptFile` скрипт, который фактически включает в себя `custom elements` polyfills в `jsdom` . Вот как выглядит наш `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);
};
```

Что делает этот скрипт, просто: удалите содержимое файлов стилей, поскольку мы не нуждаемся / не хотим его протестировать и избегаем шаблонов, когда мы импортируем их, например, с помощью синтаксиса `require('template.html')` . Затем он переносит содержимое на babel-трансформатор.

Последнее, что нужно сделать, - это включить `polyfills web components` . По умолчанию `jsdom` не поддерживает их. Для этого мы можем просто добавить `setupTestFrameworkScriptFile` в нашем примере это `jest.js` со следующим содержимым:

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

Таким образом, мы можем получить доступ к **API** `web components` в `jsdom` .

После настройки всего у нас должна быть такая структура:

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

Где мы держим наши тесты в `test` каталоге и можем запускать его с помощью команды:
`yarn run jest --no-cache --config $(node jest.config.js)` .

Прочитайте Тестирование веб-компонентов онлайн: https://riptutorial.com/ru/web-component/topic/10057/тестирование-веб-компонентов

кредиты

S. No	Главы	Contributors
1	Начало работы с веб-компонентом	Community, Mike, Supersharp
2	Тестирование веб-компонентов	Vardius