Web Components in Action

- AlexKorzhikov
- JorenBroekema

23 June 2021

Agenda

- Custom Elements
  - Standalone Elements
  - Built-in Elements
  - LifeCycle Hooks
- Shadow DOM
  - Slots
  - Styles
- HTML Template
- EcmaScript Modules

Hello Web Components

- Custom Elements - a way to create classes of elements 🏛️
- Shadow DOM - isolated DOM 🏠
- HTML Templates - standard elements to support templating functionality 🧱
- ~~HTML Imports~~ ➡ **EcmaScript Modules** - to import and export dependencies 👩‍🔧

Custom Elements

Custom elements provide a way for authors to build their own fully-featured DOM elements

A custom element is an element that is custom 🤔

Everything is a Component

For instance - **select, input & form**

```
<select>
  <option value="1">7</option>
</select>
```

Would be great to have a **multi-select element***!*

```
<multi-select>
  <option value="1">8</option>
  <option value="2">13</option>
</multi-select>
```

Example

```
<script>
class HelloWorldElement extends HTMLElement {
  connectedCallback() {
    this.textContent = "Hello World"
  }
}
customElements.define('hello-world-element', HelloWorldElement)
</script>
```

- *How can we use* **hello-world-element***?*

Declaration

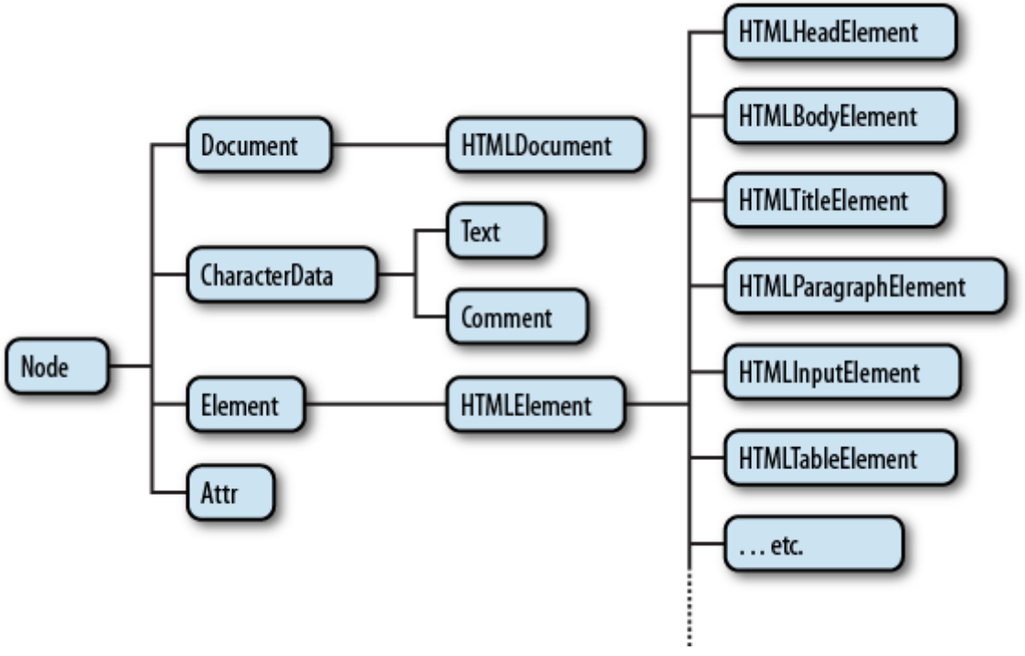
```
customElements.define("flag-icon", FlagIcon)
// [a-z](PCENChar)* '-' (PCENChar)*

// use createElement
const flagIcon = document.createElement("flag-icon")
flagIcon.country = "jp"
document.body.appendChild(flagIcon)

// use new
const flagIcon = new FlagIcon()
flagIcon.country = "jp"
document.body.appendChild(flagIcon)
// use HTML document
```

- *What is a minimal tag name to define a custom element?*

HTMLElement

- *How hard is to create a custom* **button** *element?*
- *What we need to implement?*

```
class HelloComponent extends HTMLElement { }
```



Customized Built-in Elements

- **reuse && extend** - existing behavior
- **extends && is** - required attributes

```
class PlasticButton extends HTMLButtonElement {
  constructor() {
    super() // ...
  }
}

customElements.define("plastic-button",
  PlasticButton, { extends: "button" }
)

document.createElement("button", {
  is: "plastic-button"
})
```

```
<button is="plastic-button">Click Me!</button>
```

- Demo - Make a custom label element to activate links

CustomElementRegistry

1. **window.customElements**
2. **define()**
3. **get()**
4. **whenDefined()**
5. **upgrade()**

- *What if a* **custom element** *is declared after it's being created?*

Flow

```
<example-element></example-element>
```

```
const inDocument = document.querySelector('example-element')
const outOfDocument = document.createElement('example-element')

console.assert(inDocument instanceof HTMLElement)
console.assert(outOfDocument instanceof HTMLElement)

class ExampleElement extends HTMLElement {}
customElements.define('example-element', ExampleElement)

console.assert(inDocument instanceof ExampleElement)
console.assert(!(outOfDocument instanceof ExampleElement))

// upgraded
document.body.appendChild(outOfDocument)
console.assert(outOfDocument instanceof ExampleElement)
```

LifeCycle

- **constructor** (0)
- **attributeChangedCallback** (1) <= static **observedAttributes()**
- **connectedCallback** (2) - 🏗️ **DOM**
- **disconnectedCallback** (N) - ✂️ **DOM**
- **adoptedCallback** (?) => "new document"
- Demo Execute all kinds of hooks

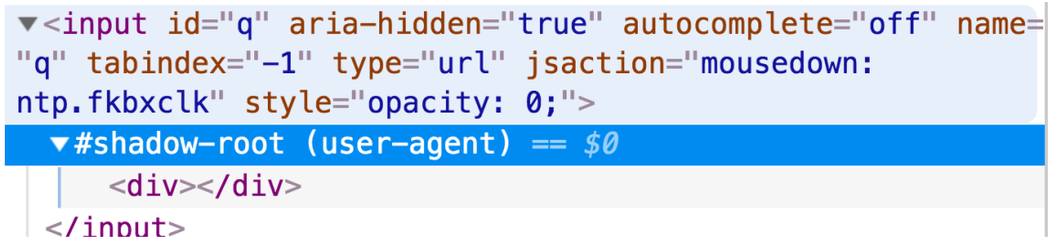
(with the help of console)

Custom Elements

Q&A

Shadow DOM

Shadow DOM fixes CSS and DOM. It introduces scoped styles to the web platform



Shadow DOM removes the brittleness of building web apps

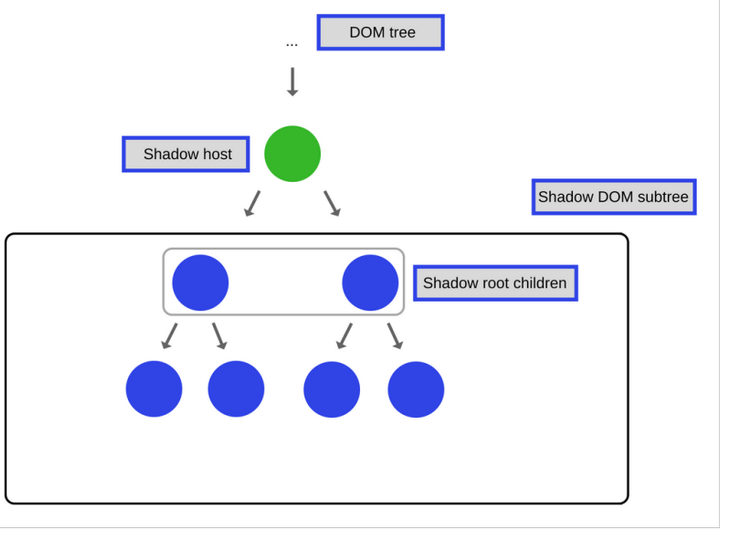
Features

- Isolated DOM - **document.querySelector()** will not work
- Composition - the component-based approach to decompose applications
- Scoped CSS - styles are not applied for document
- Simplifies CSS - use simple selectors

Almost like an iframe!

Shadow Definitions

- Tree - separate **DOM**
- Root - **Document Fragment**
- Host - **parent** element
- **mode = 'open' ll 'closed'**




```
const host = document.createElement('div')
const shadowRoot = host.attachShadow({ mode: 'open' })
shadowRoot.innerHTML = '<h1>ShadowDOM</h1>'

// host.shadowRoot === shadowRoot
// shadowRoot.host === host
// openOrCloseShadowRoot ?!
```

Shadow DOM

...a method of combining multiple DOM trees into one hierarchy and how these trees interact with each other within a document, thus enabling better composition of the DOM

© W3C

```
const host = document.createElement('div')
const shadowRoot = host.attachShadow({
  mode: 'open'
})

shadowRoot.innerHTML = `
  <style>h3{ color: red; }</style>
  <h3>Shadow DOM</h3>
`
```

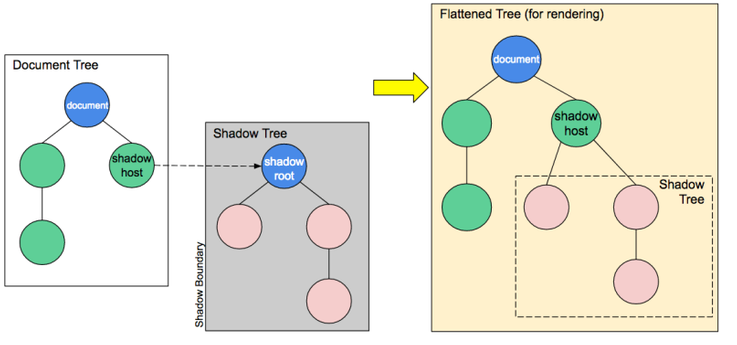
Slots

Slots are placeholders **inside your component that users can fill with their own markup**

```
<!-- defining a custom component with slots -->
<p>
  <slot name="my-text">default</slot>
</p>
<!-- using a custom component with slots -->
<my-paragraph>
  <ul slot="my-text">
    <li>different text</li>
    <li>in a list</li>
  </ul>
</my-paragraph>
```

3 DOMs

- Light - customer
- Shadow - developer
- Flattened - result



Demo - oh-my-slot

```
customElements.define('oh-my-slot', class extends HTMLElement {
  constructor() {
    super()
    this.attachShadow({
      mode: 'open'
    })
  }
  connectedCallback() {
    this.shadowRoot.innerHTML = `My Element
      <slot name="title">Default</slot>
      <slot>Default</slot>`
  }
})
```

```
<oh-my-slot>
  <h1 slot="title">Title</h1>
  <pre>Code</pre>
</oh-my-slot>
```

Styles

Shadow DOM enables describing isolated styles

```
const host = document.createElement('div')
const shadowRoot = host.attachShadow({
  mode: 'open'
})

shadowRoot.innerHTML = `
  <style>
  :host {
    display: block;
  }
  </style>
  <h3>Shadow DOM</h3>
`
```

- **:host** to describe styles applied to root component
- **:host-context** for defining context
- Styles Demo

Shadow DOM

Q&A

HTML Template

The template element is used to declare fragments of HTML that can be cloned and inserted in the document by script

© WHATWG

```
<template id="mytemplate">
  <img src="" alt="great image">
  <div class="comment"></div>
</template>
```

- **<template>** element's content is not rendered
- **document.importNode()** creates a new copy of the specified element

```
const template = document.querySelector('#mytemplate')

template.content.querySelector('img').src = 'logo.png'
const clone = document.importNode(template.content, true)
document.body.appendChild(clone)
```

Template & Shadow DOM

```
<template id="mytemplate">
  <style>
    :host {
      color: red
    }
  </style>
  My Element
</template>
```

```
customElements.define('oh-my-god', class extends HTMLElement {
  constructor() {
    super()
    this.attachShadow({
      mode: 'open'
    })
  }
  connectedCallback() {
    const template = document.querySelector('#mytemplate')
    const clone = document.importNode(template.content, true)
    this.shadowRoot.appendChild(clone)
  }
})
```

HTML Template

Q&A

Templates allow you to declare fragments of markup which are parsed as HTML, go unused at page load, but can be instantiated later on at runtime

EcmaScript Modules

Design pattern which provides the features to organize separate parts of code

```
import * as core from '@uirouter/core'
// complete module import

export default 'ui.router'
// default export

export const name = 'myName'
// named exports

import('./a').then(({ a }) => {
  console.log(a)
})
// dynamic import
```

Features

```
<script type="module" ...>
```

- Declarative
- Static declarations at the top level
- Strict mode
- Asynchronous
- Scoped
- Loaded one time!

Features

- Default & named exports & even re-export

```
export * from 'src/other_module'
```

- Imports are hoisted
- Imports are read-only views on exports

```
export let counter = 3
export function incCounter() {
    counter++
}

//------ main.js ------
import { counter, incCounter as increment } from './lib'

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

Thanks! Questions?

- *Discussed in details* **Custom Elements**, **Shadow DOM**, **EcmaScript Modules** *and* **HTML Template** *standards*
- *Show Web Components demos*

Good luck with practice!