Web Components Nativos

2021

CORPORATE UNIVERSITY

everis

WEB COMPONENTS NATIVOS

Quiénes somos

Vladimir Bataller

https://www.linkedin.com/in/vladimirbataller/

Sara Mozaffary

https://www.linkedin.com/in/saramozaffaryaliste/

Dónde estamos

T&AS

DEX

Equipo Front-tech

Vladimir

Sara

CORPORATE UNIVERSITY

everis

Indice

1. Introducción
2. Custom Elements
3. HTML Templates
4. Shadow DOM
5. Custom Properties
6. Custom Events
7. Introducción a Stencil

1. Introducción

Introducción

Qué son los Web Components

Son un conjunto de 4 especificaciones (hoy en día solamente 3, pues una está deprecada) disponibles en los navegadores actuales:

- **Custom Elements**: permite la creación de componentes reutilizables con funcionalidad propia, que se incluyen en un documento HTML mediante una etiqueta personalizada.
- **HTML Templates**: define plantillas HTML/CSS inicialmente ocultas que pueden ser renderizadas en el punto del DOM que se desee.
- **Shadow DOM**: API JavaScript que permite insertar en el DOM elementos encapsulados que no son accesibles desde fuera ni por JavaScript (mediante búsquedas del tipo getElementById, querySelector, etc), ni mediante estilos externos al Web Component.
- ~~**HTML Imports**~~: (actualmente deprecado). Permitía importar HTML mediante un link rel=”import” href=”archivo_importado.html”.

Introducción

Pero… ¿Qué son los Web Components?

Cuando se habla de Web Components normalmente se hace referencia a una tecnología nativa que permite crear componentes:

- Encapsulados*
- Autocontenidos
- Reutilizables

Introducción

Aproximación básica

La aproximación básica para implementar un componente web generalmente es la siguiente:

- Crear una clase en la que especificar su funcionalidad.
- Registrar el elemento.
- Adjuntar un shadow DOM.
- Definir una plantilla HTML.
- Insertar el web component.

Introducción

Preparación del espacio de trabajo

Clonar el repositorio con los ejemplos:

```
git clone https://github.com/vladimirbat/curso_webcomponents_nativos.git
```

Hacer una copia del código fuente en otra carpeta para poder hacer los ejemplos por tu cuenta.

2. Custom Elements

Custom Elements

Definición

Es el API que, de forma nativa para el navegador, permite la creación **componentes reutilizables**. Estos componentes tienen funcionalidad propia, que se incluyen en un documento HTML mediante una **etiqueta** personalizada asociada a dicho componente.

https://developer.mozilla.org/es/docs/Web/Web_Components/Using_custom_elements

Custom Elements

Clase que implementa la funcionalidad

Se declaran como una clase JS que hereda de **HTMLElement**.

La primera línea del constructor debe ser una llamada al constructor de la superclase (*super()*).

El propio Custom Element debe insertarse (appendChild) los elementos HTML que renderizará en su interior.

```
class HolaMundoComponent extends HTMLElement{
    constructor(){
        super();
        this.insertado = false;
    }
    connectedCallback(){
        if(this.isConnected && !this.insertado){
            this.insertado = true;
            const div = document.createElement('div');
            div.className = 'saludo'
            div.textContent = '¡Hola Mundo!';
            this.appendChild(div);
        }
    }
}
```

Custom Elements

Declaración del Custom Element

Mediante el método ***define*** del objeto ***window.customElements***, se declara públicamente en el documento HTML el Custom Element. Este método recibe dos argumentos:

- Etiqueta que se empleará para insertar el Custom Element en el HTML de la página. ***Esta etiqueta debe declararse en minúsculas y con al menos un guión***.
- Clase JS que implementa la funcionalidad del Custom Element.

```
customElements.define('hola-mundo',HolaMundoComponent);
```

Custom Elements

Inserción del Custom Element en el documento HTML

Por último, el Custom Element se inserta en el documento HTML mediante el nombre de etiqueta declarado en el método ***define***.

```
<h1>Custom Element:</h1>
<hola-mundo></hola-mundo>
```

Custom Elements

Atributos de entrada (I)

```
connectedCallback(){
    if(this.isConnected && !this.insertado){
      this.insertado = true;
      const div = document.createElement('div');
      div.className = 'saludo'
      const nombre = this.getAttribute('nombre');
      const apellidos = this.getAttribute('apellidos');
      div.textContent = `¡Hola ${nombre} ${apellidos}!` ;
      this.appendChild(div);
    }
}
```

Un Custom Element puede recibir datos del exterior (resto del documento HTML) mediante atributos (o propiedades).

El Custom Element, esos atributos se pueden leer mediante el método ***getAttribute*** que recibe como argumento el nombre del atributo.

Custom Elements

Atributos de entrada (II)

Los valores de los atributos se le pueden pasar al componente directamente en el HTML.

En este ejemplo, los atributos solamente se pueden leer si están asignados a la etiqueta desde el principio. Para leer los valores asignados desde JavaScript una vez el componente se ha visualizado, se debe emplear la detección de cambios explicada en el siguiente apartado.

```
<h1>Custom Element:</h1>
<hola-mundo nombre="Daniel" apellidos="Valiente"></hola-mundo>
<hr/>
<hola-mundo nombre="Laura" apellidos="Villanueva"></hola-mundo>
```

Custom Elements

Detección de cambios de los atributos de entrada

Se puede indicar los atributos de entrada cuyos cambios se quieren observar. Para ello, el método ***static get observedAttributes()*** debe retornar un array con los nombres de los atributos observados.

```
static get observedAttributes () {
    return [ 'num' ];
}
```

El método (*callback*) que se ejecutará cuando dichos atributos cambien será *attributeChangedCallback*:

```
attributeChangedCallback (attrName, oldValue, newValue) {
    console.log(attrName + ':',  oldValue , '->',  newValue);
}
```

Custom Elements

CORPORATE UNIVERSITY
everis
WEB COMPONENTS NATIVOS

Métodos del ciclo de vida de los Custom Elements

Constructor -> inicialización de variables

connectedCallback() -> Cuando se inserta el Custom Element en el DOM (se puede ejecutar varias veces si se quita y se inserta en el DOM). Para saber en cualquier momento si el Custom Element está conectado se debe leer el atributo ***isConnected***.

disconnectedCallback() -> Cuando se quita el Custom Element del DOM, es el momento adecuado para liberar recursos, por ejemplo quitar eventos (removeEventListener).

attributeChangedCallback(attrName, oldValue, newValue) -> Cuando el atributo cuyo nombre se recibe ha cambiado. Solamente los incluidos en el array retornado por el método estático *get observedAttributes* lanzan el evento *attributeChangedCallback*.

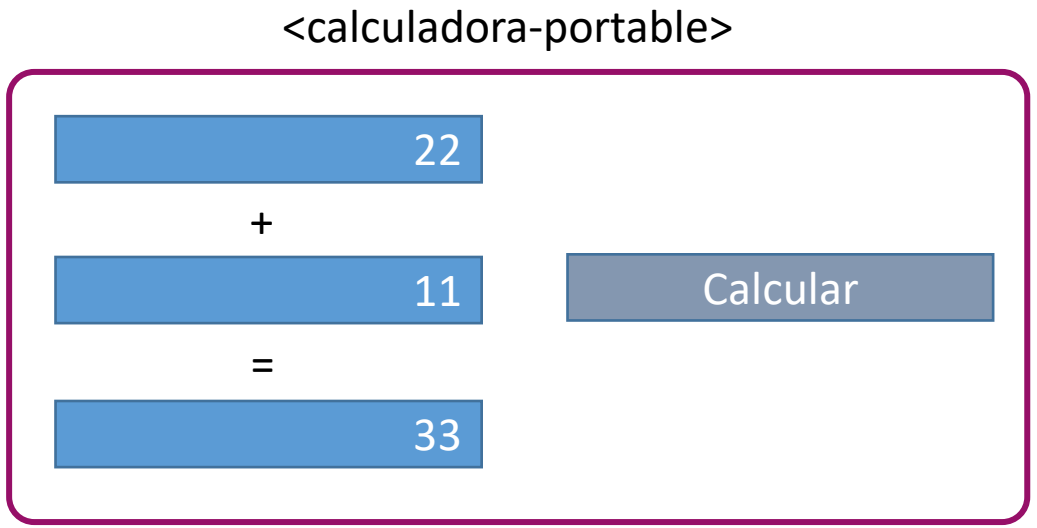
adoptedCallback() -> Cuando el CustomElement es adoptado por otro document (ejecución de *document.addoptNode(element)*), cosa que se hace cuando se comparten nodos entre un *iframe* y su *document* contenedor.

Custom Elements

Práctica I

Crear un Custom Element que implemente una calculadora. Tendrá un atributo de entrada que indicará la operación a realizar (+, -, * o /).

Visualmente mostrará dos cajas de texto para insertar dos números. Entre las dos cajas de texto mostrará la operación que se va a realizar (insertada como atributo) y además mostrará un botón para ejecutar la operación y abajo del todo mostrará el resultado de la operación.



3. HTML Templates

HTML Templates

Etiqueta *template*

La etiqueta **template** permite contener un fragmento HTML que no es renderizado por el navegador. Esto constituye un mecanismo de plantillas de código cuyo contenido (Fragmento HTML) puede ser insertado (y renderizado), cuando sea necesario, en el punto del DOM que se desee.

```
<template id="plantilla">
    <div>
        <ul>
            <li>A</li>
            <li>B</li>
        </ul>
    </div>
</template>
<div><button id="mostrarPlantilla">Mostrar Plantilla</button></div>
<div id="marco">

</div>
```

HTML Templates

Obtener, clonar e insertar una *template*

- Obtener (con *querySelector*, *getElementById*, etc) una referencia al elemento *template* que se desea insertar en el DOM.
- Mediante el atributo ***content*** obtener el fragmento del DOM que contiene.
- Clonar dicho fragmento, con todos sus hijos, e insertarlo en el punto del DOM donde se quiera renderizar la plantilla.

```
document.querySelector('#mostrarPlantilla').addEventListener('click', () => {
    const plantilla = document.querySelector("#plantilla");
    const content = plantilla.content;
    const fragment = content.cloneNode(true);
    const destino = document.querySelector('#marco');
    destino.appendChild(fragment);
});
```

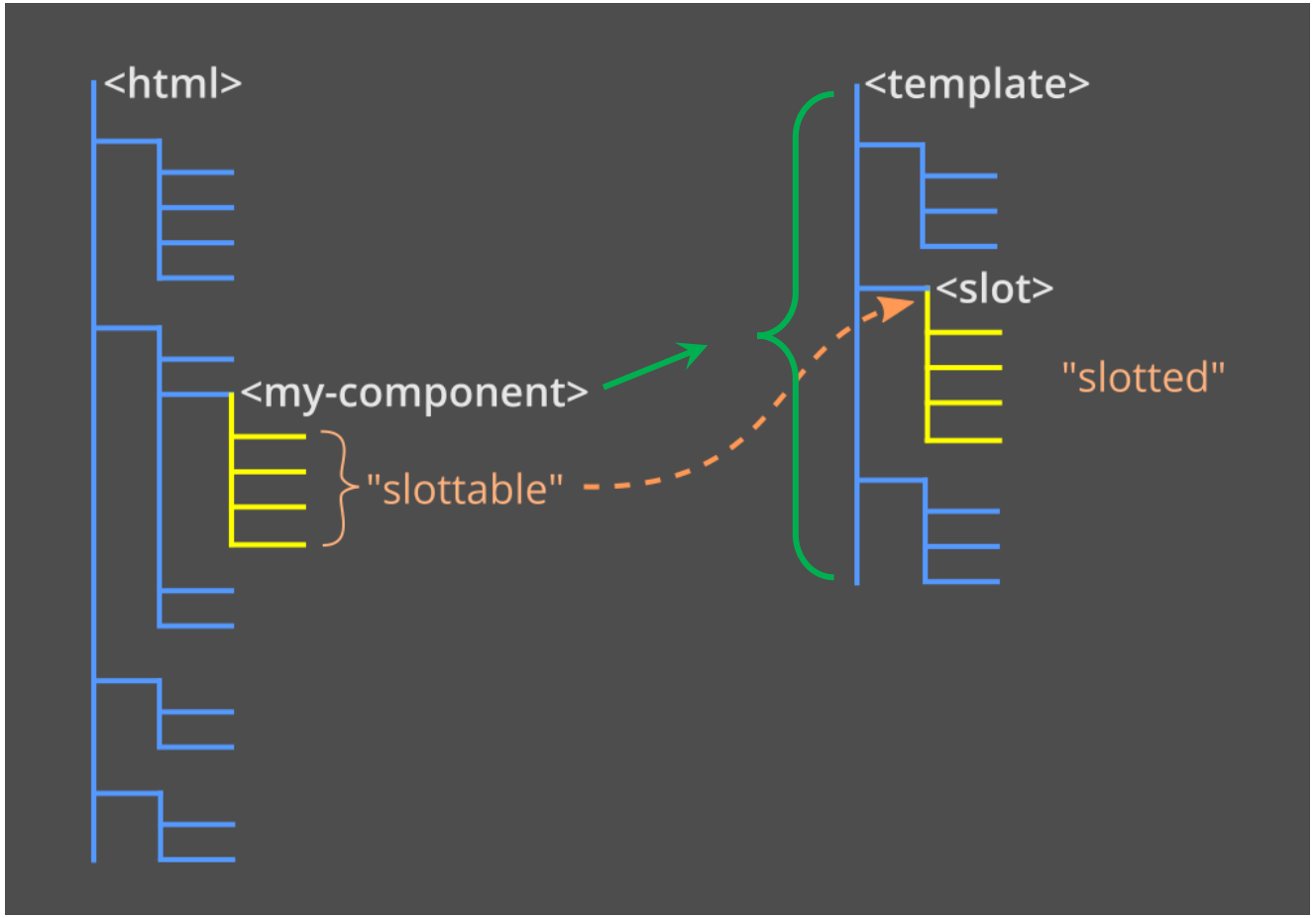
HTML Templates

CORPORATE UNIVERSITY
everis
WEB COMPONENTS NATIVOS

Uso de *HTML Template* con Custom Elements

Las plantillas HTML adquieren mayor potencia en su uso en Custom Elememnts, pues se pueden particularizar con:

- Slots (etiqueta slot)
- Estilos encapsulados dentro del Custom Element.



HTML Templates

Slots: plantilla con slot

A continuación se muestra una plantilla con un slot definido (con nombre texto-interior) que se insertará dentro de un Custom Element.

```
<template id="plantilla">
    <style>
        p {border: 1px solid black;box-shadow: 10px 5px 5px gray;padding: 2em;}
    </style>
    <p><slot name="texto-interior">Mi texto predeterminado</slot></p>
</template>
```

HTML Templates

Slots: dar contenido al slot

Cuando se haga uso del Custom Element (en el ejemplo *borde-con-sombra*), en su interior se puede incluir el fragmento que se quiere insertar en el slot cuyo nombre se indica con el atributo ***slot*** (en este caso el nombre del slot es '*texto-interior*').

```
<borde-con-sombra>
    <div slot="texto-interior">
        <ul>
            <li>¡Lunes!</li>
            <li>¡Martes!</li>
        </ul>
    </div>
</borde-con-sombral>
```

HTML Templates

Slots: declaración del Custom Element

A continuación se muestra el código del Custom Element con el slot mostrado anteriormente.
Como se puede apreciar, no es necesario agregar ninguna referencia al slot.

```
customElements.define('borde-con-sombra',
    class extends HTMLElement {
        constructor() {
            super();
            const plantilla = document.getElementById('plantilla');
            this.attachShadow({mode: 'open'}).appendChild(plantilla.content.cloneNode(true));
        }
    }
);
```

Nota: para que los Slots funcionen, la plantilla se debe insertar en un Shadow DOM.

HTML Templates

CORPORATE UNIVERSITY
everis
WEB COMPONENTS NATIVOS

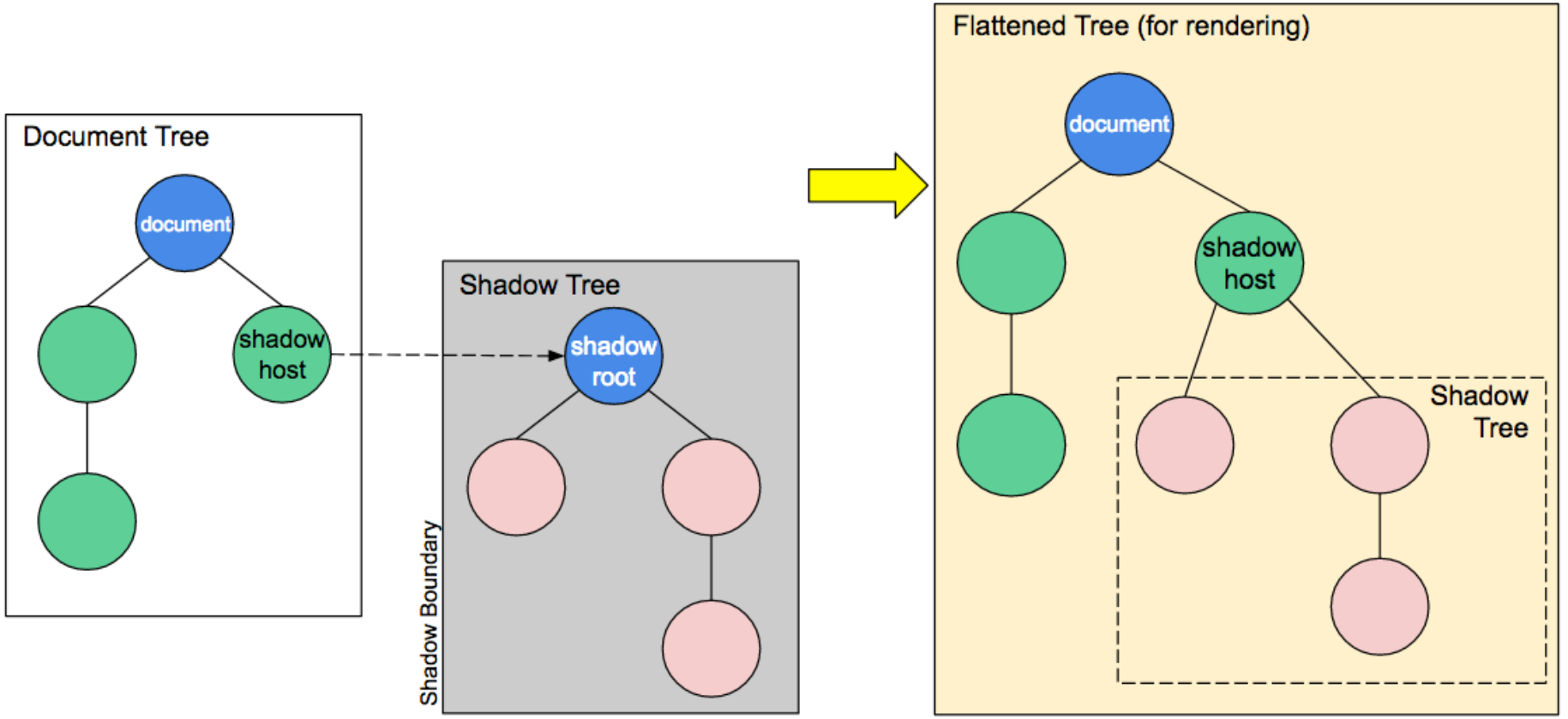
Slots: particularidades

- Para que los Slots funcionen, la plantilla se debe insertar en un Shadow DOM.
- Los estilos interiores al Custom Element (con su Shadow DOM), no afectan al fragmento recibido en un slot.
- El fragmento recibido en un Slot, tiene los estilos que se le aplican desde fuera del Custom Element.
- Un Custom Element puede tener varios Slots (cada uno con su nombre).

4. Shadow DOM

Shadow DOM

Topología del Shadow DOM



Shadow DOM

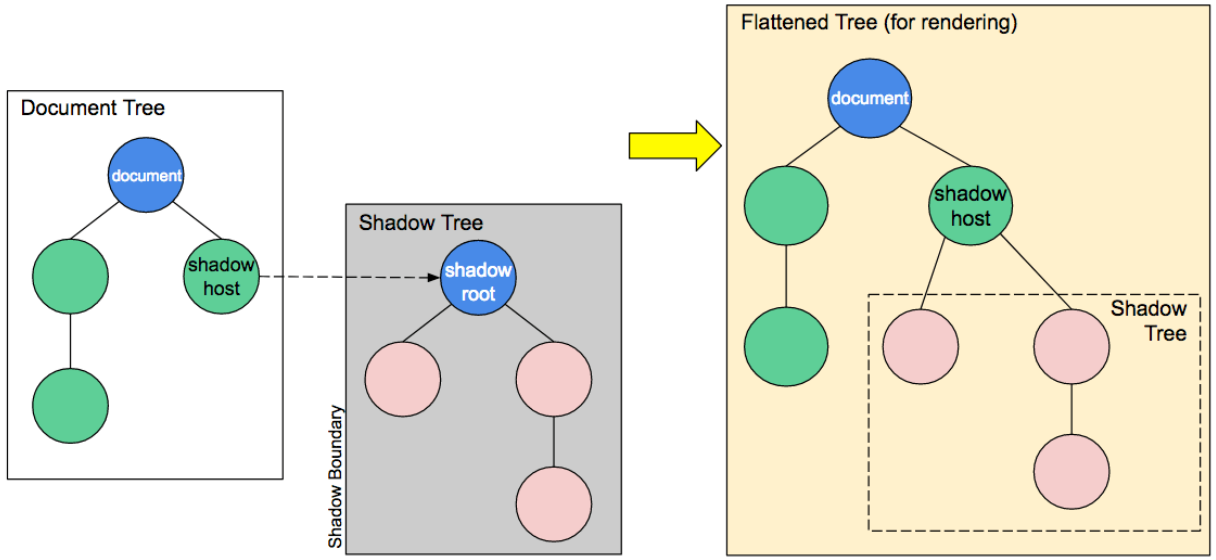
Definiciones

Shadow host: nodo del DOM al que es atado el Shadow DOM.

Shadow tree: árbol DOM dentro del shadow DOM.

Shadow boundary: donde el shadow DOM termina y el DOM regular comienza.

Shadow root: nodo raíz del árbol Shadow.



Shadow DOM

Nivel de acceso al DOM y estilos de un Custom Element

A continuación se muestran los diferentes niveles de acceso al DOM y los estilos de Custom Elements con/sin Shadow DOM y con tipo open/closed.

	Sin Shadow DOM	Shadow DOM open	Shadow DOM closed
Acceso al DOM del WC desde fuera	Si	Mediante propiedad ***shadowRoot***	No
Estilos exteriores	Si	No	No
Custom properties	Si	Si	Si

Shadow DOM

Creación y variable de acceso a un Shadow DOM

El método ***attachShadow*** del Custom Element permite asignarle un Shadow DOM. Este método recibe un objeto ***options*** de configuración que puede tener los siguientes argumentos:

Atributo	Descripción
mode	Puede tomar los valore ‘open’ o ‘closed’ para Indicar si el Shadow DOM está abierto o cerrado.
delegatesFocus	Por defecto a false y permite indicar si el propio Shadow DOM puede recibir el foco, esto puede ser útil para estilos con selector :focus que se quieran aplicar.

Shadow DOM

Creación y variable de acceso a un Shadow DOM

El método ***attachShadow*** retorna una referencia al Shadow DOM creado, si este es de tipo 'closed' esta será la única forma de acceder a dicho Shado DOM.

Si el Shadow DOM creado es 'open', también existe el atributo shadowRoot accesible internamente mediante this.shadowRoot y externamente mediante una referencia al Custom Element obtenida con getElementbyId, querySelector, etc.

```
const shadowRoot = this.attachShadow({mode: 'open'});
shadowRoot.appendChild(plantilla.content.cloneNode(true));
```

Shadow DOM

Estilos dentro del Shadow DOM (I)

Los estilos dentro del Shadow DOM se pueden aplicar:

- Mediante una plantilla que tenga un **elemento *style***.
- Ejecutando un appendChild de un elemento style.
- Mediante un archivo de estilos externo importado con @import dentro de un elemento style:

```
<template id="templateContent">
     <style> @import "css/generalStyle.css"; </style>
</template>
```

Shadow DOM

Estilos dentro del Shadow DOM (II)

Dentro del Shadow DOM, además de los selectores de estilo estándar, se pueden emplear el selector :host para caracterizar estados del elemento del Custom Element.

```
:host {
  opacity: 0.4;
  transition: opacity 420ms ease-in-out;
}
:host(:hover) {
  opacity: 1;
}
:host(:active) {
  position: relative;
  top: 3px;
  left: 3px;
}
```

Nota: El selector css /deep/ que permitía acceder a estilos dentro del Shadow DOM, ha sido deprecado.

5. Custom Properties

Concepto y uso en Web Components

Custom Properties

Custom Properties: declaración

Las Custom Properties se declaran dentro de un ámbito css, este puede ser un selector específico y las variables solamente se aplicarán cuando ese selector se cumpla o dicho ámbito puede ser :root para que se apliquen a toda la página.

```
div.resaltado {
    --main-bg-color: brown;
}
:root {
    --main-bg-color: gray;
}
```

Custom Properties

Custom Properties: lectura

Para leer en un bloque CSS el valor de una Custom Property, se emplea la función ***var()*** que recibe el nombre de la Custom Property (incluyendo los dos guiones)

```
div.resaltado {
    background-color: var(--main-bg-color);
}
p.coloreado {
    background-color: var(--main-bg-color);
}
```

6. Custom Events

Custom Events

Introducción

Los Custom Events permiten crear eventos equivalentes a los eventos nativos del navegador, pero asociados a la funcionalidad de una aplicación o un Custom Element. Para ello, el primer paso es crear una instancia del Custom Event que se quiere emitir.

```
const CalculadoraEvent = new CustomEvent('calculadoraEvent', options);
```

Para lanzar un evento, sobre el elemento que lo emite se debe ejecutar el método dispatchEvent pasandole el objeto CustomEvent que se quiere emitir.

```
const cancelled = this.dispatchEvent(CalculadoraEvent)
```

Para para suscribirse al evento, sobre una referencia del objeto que lo emite, se debe ejecutar el método addEventListener indicando el nombre del evento.

```
element.addEventListener('calculadoraEvent', (event)=>{});
```

Custom Events

Creación de un Custom Event

En la creación de un Custom Event, el primer argumento es el nombre por el que se identificará el evento y el segundo es un objeto con las opciones de configuración del Cutom Event.

```
const CalculadoraEvent = new CustomEvent('calculadoraEvent', options);
```

El objeto ***options*** puede recibir los siguientes atributos:

Atributo	Descripción
detail	Objeto que permite pasar datos asociados al evento y dependientes de la funcionalidad.
bubbles	Si es true el evento se debe propagar en la fase de burbuja (de elementos hijos a padres).
composed	Si es true el evento se puede atravesar en Shadow DOM.
cancelable	Si es true el evento se puede cancelar en uno de los manejadores con ***.preventDefault()***

Custom Events

Emisión y cancelación de eventos

Los Custom Events se emiten con el método ***dispatchEvent*** del objeto que lo emite (this si es un Custom Element). Este método retorna un booleano que indica si el evento ha sido cancelado por uno de sus receptores, lo que permite ejecutar o no la acción asociada al evento.

```
const cancelled = this.dispatchEvent(CalculadoraEvent);
if(!cancelled) {
    this.navigate();
}
```

En la suscripción al evento, para cancelar la acción asociada al evento, se puede ejecutar sobre el objeto evento recibido el método ***preventDefault***.

```
element.addEventListener('calculadoraEvent', (event)=>{
    event.preventDefault();
    console.log('El receptor ha cancelado la acción del evento');
});
```

CORPORATE
UNIVERSITY

everis