

Web Components

Elementos Web personalizados
y reutilizables

Tony G. Bolaño

tony@tonygb.com

KeepCoding Full Stack Web Developer

Marzo 2021





■ ¿Qué son los web components?



■ Definición

- Conjunto de diferentes tecnologías
- Permiten crear elementos personalizados reutilizables
- Funcionalidad encapsulada

OBJETIVO:

Crear componentes para una experiencia de usuario de alta calidad de forma nativa, sin complicados frameworks, procesos complicados de compilación ni riesgo de quedar obsoletos.



Tecnologías principales

- **Custom elements**: Elementos HTML personalizados con su propia etiqueta plantilla, comportamiento.
(Estándar HTML: <https://html.spec.whatwg.org/multipage/custom-elements.html#custom-elements>)
- **Shadow DOM**: Permite aislar los estilos CSS y el comportamiento JS.
(Estándar DOM: <https://dom.spec.whatwg.org/#shadow-trees>)
- **HTML Templates**: Plantillas HTML que no se renderizan hasta que son requeridas.
(Estándar HTML: <https://html.spec.whatwg.org/multipage/scripting.html#the-template-element>)
- **ES Modules**: Estándar para la reutilización modular y eficiente de archivos JavaScript.
(ES6 Modules en MDN: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules>)





■ Custom elements




Compatibilidad

Custom Elements (V1) - LS

One of the key features of the Web Components system, custom elements allow new HTML tags to be defined.

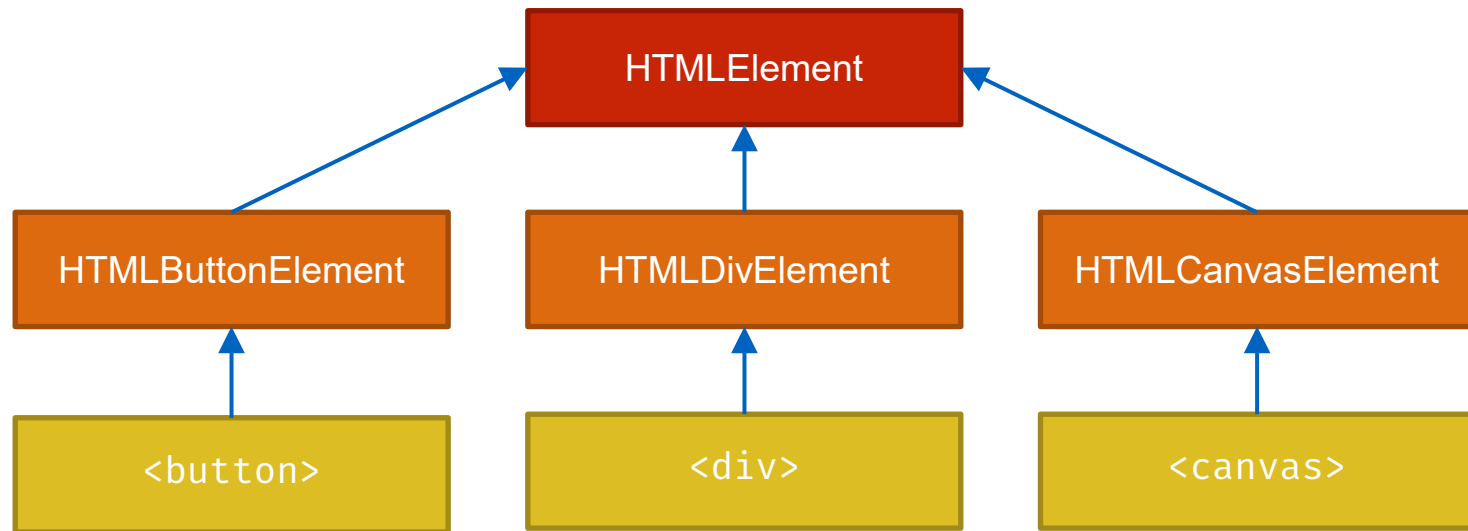
Usage	% of all users		
• Spain	79.15%	+	16.87% = 96.02%
Global	75.81%	+	18.33% = 94.13%

Current aligned Usage relative Date relative Filtered All 

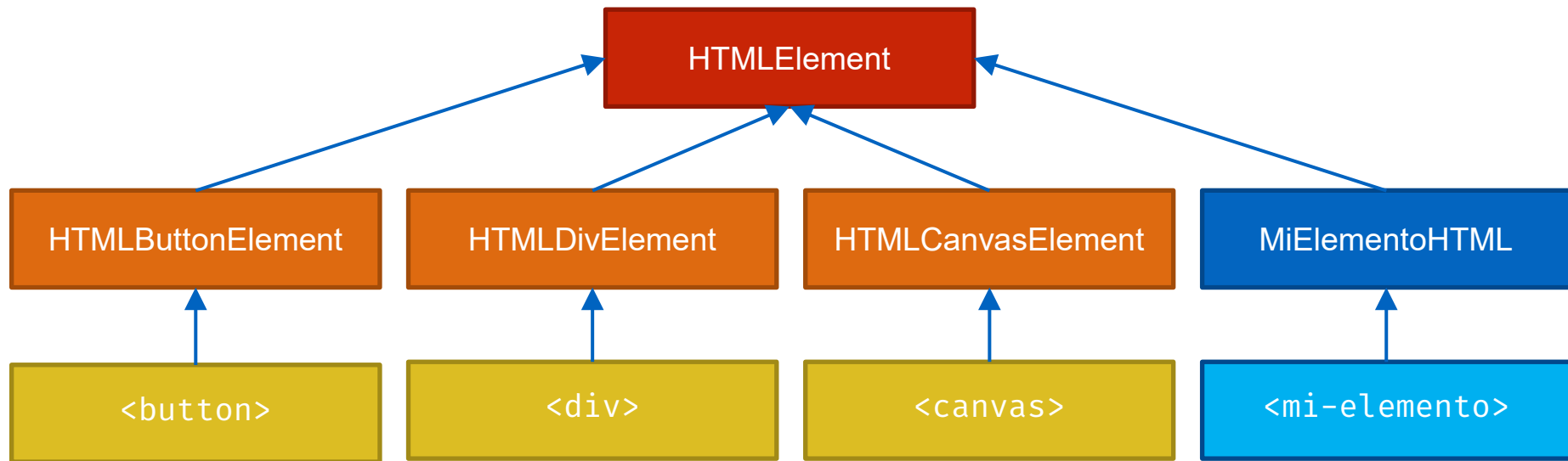
IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baidu Browser	KaiOS Browser
		2-49														
		2 1 50-58	4-53		10-40								4			
	12-18	3 1 59-62	1 54-66	3.1-10	1 41-63	3.2-10.2							1 5.4			
6-10	79-88	63-85	67-88	1 10.1-13.1	64-72	1 10.3-13.7		2.1-4.4.4	12-12.1				6.2-12.0			
11	89	86	89	1 14	73	1 14.4	all	81	62	88	86	12.12	13.0	1 10.4	1 7.12	2.5
		87-88	90-92	1 TP												



Herencia en elementos HTML



Herencia en elementos HTML



Cualquier etiqueta no reconocida por el navegador será tratada como un `<div>`



■ Cómo definir un custom element

```
class HolaComponent extends HTMLElement {  
  connectedCallback() {  
    this.innerHTML = `

# Hola Mundo</h1>`; } }


```

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

Definir una nueva clase JavaScript con la funcionalidad.

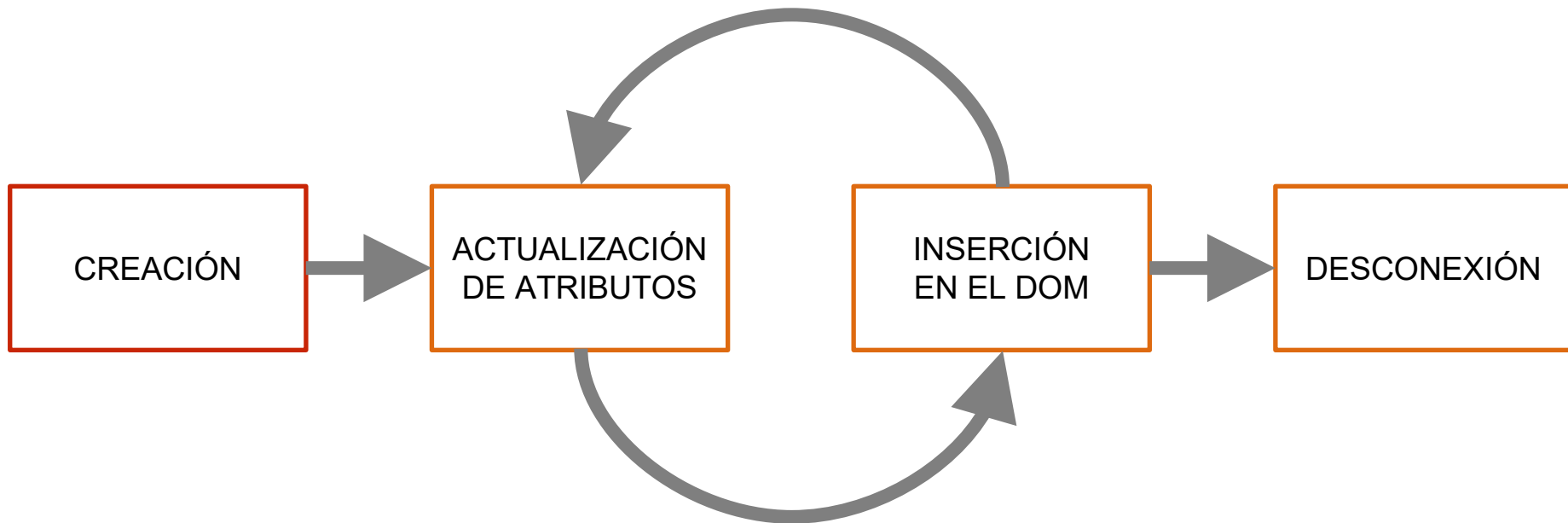
Registrar una nueva etiqueta asociando la funcionalidad.

```
<hola-mundo></hola-mundo>
```

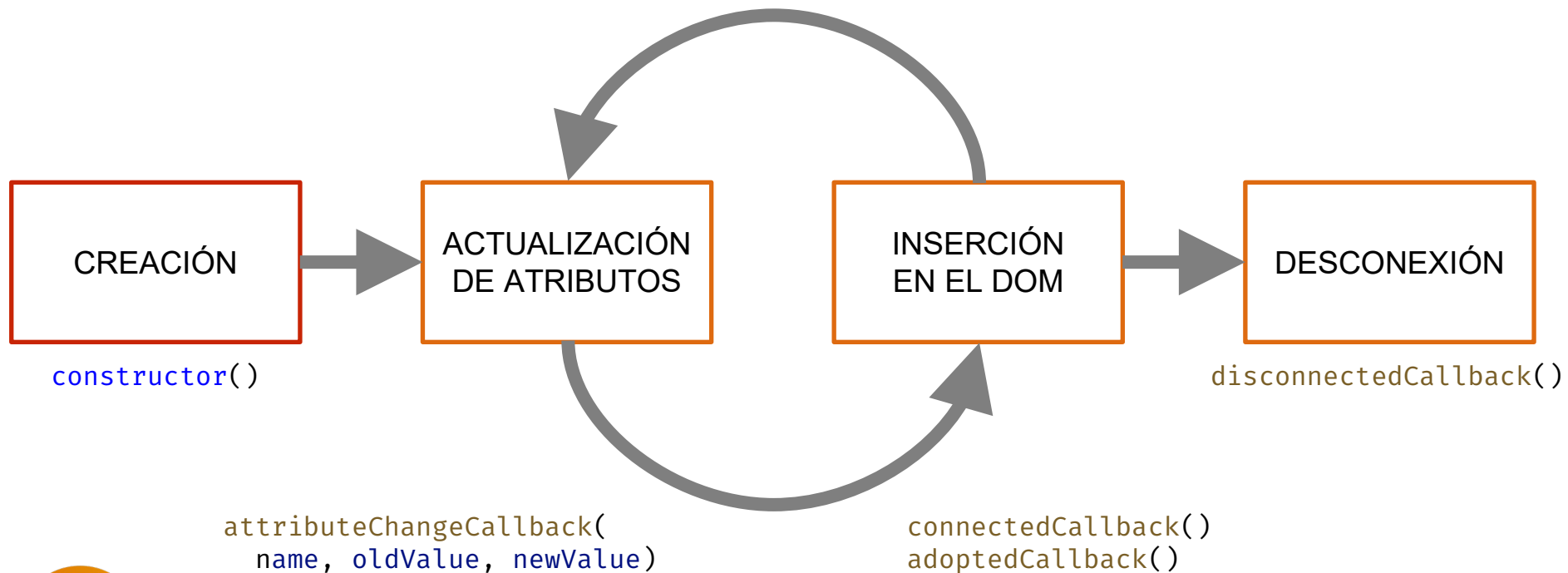
Usar el nuevo componente insertando la etiqueta en nuestro archivo HTML.



■ Ciclo de vida y hooks



■ Ciclo de vida y hooks



■ constructor()

- Obligatorio para Custom Elements (opcional en clases ES6)
- Primera sentencia: llamada a `super()` para asegurar que el componente hereda la cadena de prototipos, propiedades y métodos que extiende.
- No se puede usar `return` (salvo que sea `return` o `return this`)
- No se puede usar `document.write()` o `document.open()`
- Los atributos y elementos hijos no pueden inspeccionarse (todavía no están en el DOM)
- Puede usarse para crear un shadow DOM, suscribirse a eventos o establecer el estado del componente, pero no se recomiendan tareas como renderizar o lanzar peticiones de recursos (ver `connectedCallback`)



■ connectedCallback()

- Se lanza cuando un elemento es agregado al DOM
- Ya se pueden consultar atributos, obtener recursos, renderizar plantillas... de forma segura
- **¡CUIDADO!** Puede lanzarse **más de una vez** durante la vida de un componente



■ disconnectedCallback()

- Se lanza cuando el elemento es eliminado del DOM
- Es el sitio ideal para hacer limpieza y liberar recursos:
 - Desuscripciones de eventos del DOM
 - Parar timers
 - Eliminar los callbacks globales o de servicios de aplicación
- **NOTA:** no se ejecuta cuando un usuario simplemente cierra la ventana (o la pestaña del navegador). Puede ejecutarse **más de una vez** a lo largo del ciclo de vida.



■ attributeChangedCallback(name, oldVal, newVal)

- Responde a los cambios en los atributos del elemento
- Solamente reacciona a los cambios en los atributos especificados en el getter estático `observedAttributes`
- Parámetros:
 - Nombre del atributo
 - Valor antes del cambio
 - Valor después del cambio
- Operaciones: agregación, eliminación, actualización, reemplazo



■ adoptedCallback()

- Caso muy específico: `document.adoptNode(element)`
- En general, uso de `<iframe>`
- **NOTA:** el elemento no se destruye ni se crea en el proceso de *adopción*. Es decir, no se ejecuta el constructor.





■ Shadow DOM



Compatibilidad

Shadow DOM (V1) 📄 - WD

Method of establishing and maintaining functional boundaries between DOM trees and how these trees interact with each other within a document, thus enabling better functional encapsulation within the DOM & CSS.

Usage

% of all users

• Spain	81.29%	+ 14.8%	= 96.09%
Global	80.43%	+ 13.82%	= 94.25%

Current aligned

Usage relative

Date relative

Filtered

All



IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baidu Browser	KaiOS Browser
		2-57														
		2 58											4			
	12-18	3 59-62	4-52	3.1-9.1	10-39	3.2-9.3							5.4			
6-10	79-88	63-85	53-88	10-13.1	40-72	10-13.7		2.1-4.4.4	12-12.1				6.2-12.0			
11	89	86	89	14	73	14.4	all	81	62	88	86	12.12	13.0	10.4	7.12	2.5
		87-88	90-92	TP												



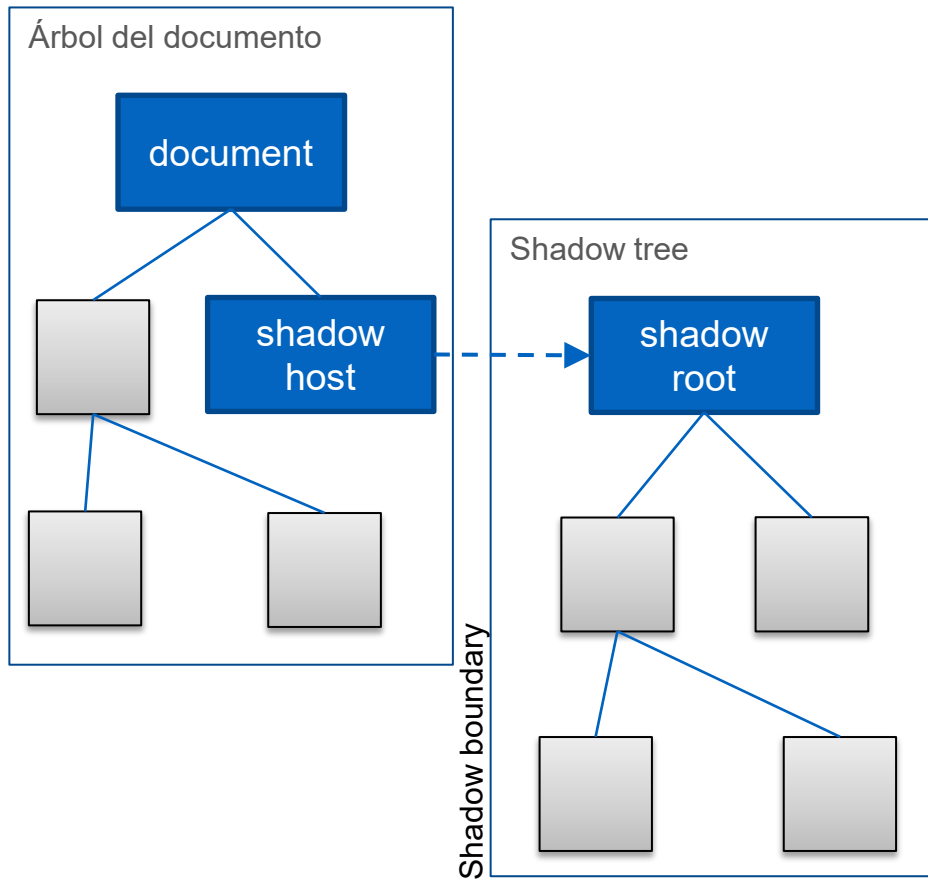
■ Shadow DOM

- Se utiliza para componentes “autocontenidos”
- Permite encapsular estilos y marcado
- Se crea mediante `element.attachShadow({ mode: 'open' })`
- Crea una raíz o `shadowRoot` a la que podemos referenciar y con la que podemos interactuar
- Permite adjuntar árboles DOM ocultos a elementos en el árbol DOM regular (light DOM)



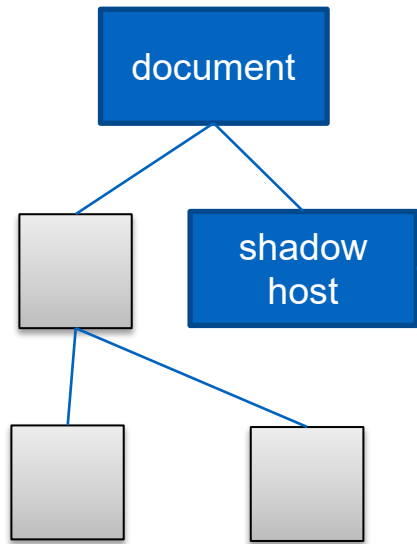
■ Conceptos del Shadow DOM

- **Shadow host:** nodo regular del DOM al que se vincula el shadow DOM.
- **Shadow tree:** árbol DOM dentro del shadow DOM.
- **Shadow boundary:** El punto en el que el shadow DOM termina y el DOM regular comienza.
- **Shadow root:** nodo raíz del árbol shadow (shadow tree).

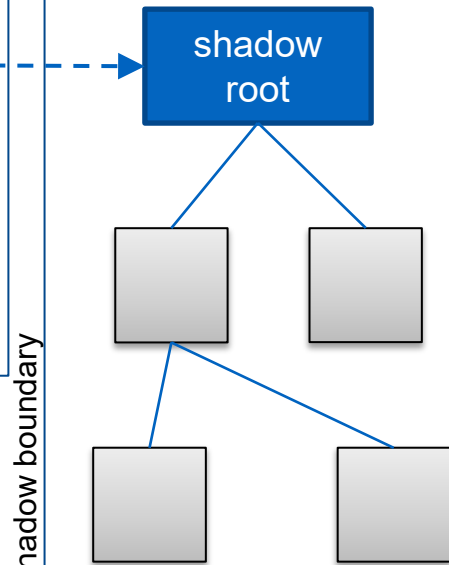


■ Conceptos del Shadow DOM

Árbol del documento

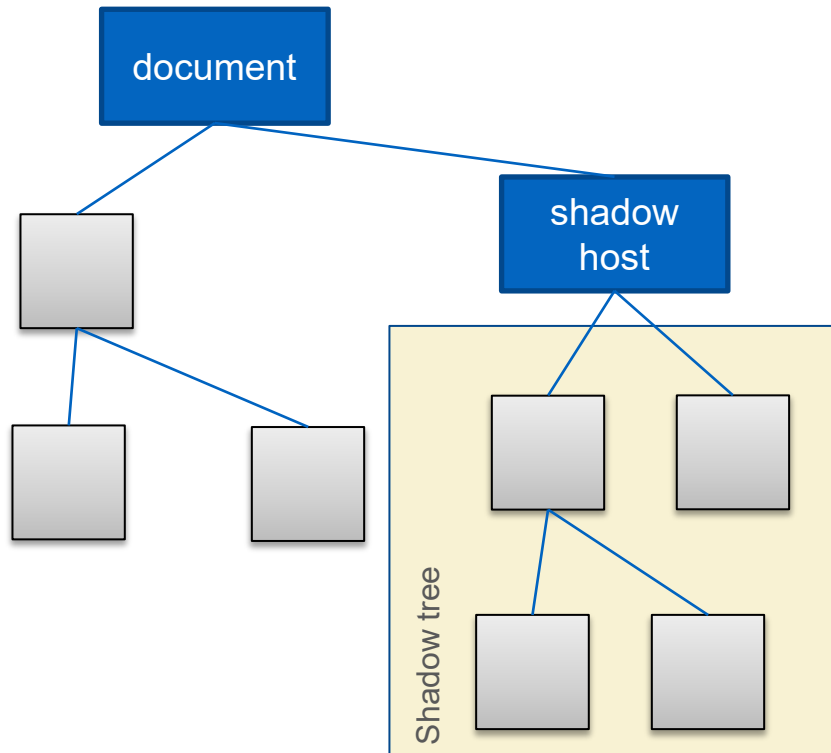


Shadow tree



Shadow boundary

Renderizado por el navegador



■ ¿Cómo se crea un shadow DOM?

```
let shadow = elementRef.attachShadow({mode: 'open'});  
let shadow = elementRef.attachShadow({mode: 'closed'});
```

- **open:** podemos acceder a los elementos del shadow DOM usando JavaScript desde el contexto principal de la página.
- **close:** impide el acceso al shadow DOM desde el contexto principal, es decir, `element.shadowRoot` devuelve `null`.
- Una vez creado, podemos agregar elementos usando la API del DOM

```
var para = document.createElement('p');  
shadow.appendChild(para)
```



Shadow DOM y Custom Elements

- Normalmente crearemos el shadow DOM en el constructor
- Siempre llamar a super()
- Podemos usar la propiedad innerHTML

```
class TgbCustomElement extends HTMLElement {  
  constructor() {  
    super();  
    this.attachShadow({ mode: 'open' });  
    this.shadowRoot.innerHTML = '<h1>Hello, Shadow DOM</h1>';  
  }  
}
```





■ HTML Templates




Compatibilidad

HTML templates - LS

Method of declaring a portion of reusable markup that is parsed but not rendered until cloned.

Current aligned Usage relative Date relative Filtered All 

Usage

% of all users  ?

• Spain 96.99% + 0.03% = 97.02%
Global 96.02% + 0.34% = 96.36%

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baidu Browser	KaiOS Browser
				3.1-6												
				6.1												
	12		4-25	7	10-12.1	3.2-7.1										
	13-14	2-21	26-34	7.1-8	15-21	8-8.4		2.1-4.3								
6-10	15-88	22-85	35-88	9-13.1	22-72	9-13.7		4.4-4.4.4	12-12.1				4-12.0			
11	89	86	89	14	73	14.4	all	81	62	88	86	12.12	13.0	10.4	7.12	2.5
		87-88	90-92	TP												



■ El elemento <template>

- Permiten usar estructuras repetidamente de forma fácil.
- El elemento <template> está soportado por los navegadores actuales
- El contenido no se representa directamente, pero puede ser referenciado por JavaScript

```
<template id="plantilla">  
  <h1>Hola, templates!</h1>  
</template>
```

```
let template = document.getElementById('plantilla');  
let templateContent = template.content;  
document.body.appendChild(templateContent);
```



■ El elemento <template>

- Permiten usar estructuras repetidamente de forma fácil.
- El elemento <template> está soportado por los navegadores actuales
- El contenido no se representa directamente, pero puede ser referenciado por JavaScript

```
<template id="plantilla">  
  <h1>Hola, templates!</h1>  
</template>
```

```
let template = document.getElementById('plantilla');  
let templateContent = template.content;  
document.body.appendChild(templateContent);
```

Aquí la plantilla se queda “vacía” y, por tanto, no podemos reutilizarla



¿Por qué usar templates?

```
<template id="plantilla">
  <h1>Hola, templates!</h1>
</template>
```

```
let template = document.getElementById('plantilla');
let templateContent = template.content;
document.body.appendChild(templateContent);
```

```
const template = document.getElementById('plantilla');
const node = document.importNode(template.content, true);
document.body.appendChild(node);
```

Realiza un *deep clone*

¡Importante! Clonar el contenido, no la plantilla





GRACIAS

www.keepcoding.io

