cedia | TMO Training Management Office

info@cedia.org.ec
cedia.edu.ec

CURSO DE JAVASCRIPT

Corporación Ecuatoriana para el Desarrollo de la Investigación y la Academia

Ing. María Fernanda Granda J., PhD

3-12-2025

"Conectando ideas, transformando sociedades"

Módulo 7: Trabajar con el Modelo Objeto Documento (DOM)

- Incompatibilidades de los navegadores y código cross-browser.

Incompatibilidades de los navegadores

- Surgen porque cada navegador interpreta el código JavaScript de manera diferente, especialmente con versiones antiguas o características no estandarizadas.
- Para abordar esto, se utiliza código cross-browser, que incluye:
  - técnicas como la detección de características,
  - el uso de polyfills y la escritura de código que maneje las diferencias entre navegadores.

¿Qué es un polyfill en JavaScript?

- Es un fragmento de código, generalmente en JavaScript, que proporciona funcionalidad moderna en navegadores antiguos que no la admiten de forma nativa.
- Los polyfills "rellenan los huecos" permitiendo a los desarrolladores usar nuevas características de JavaScript sin preocuparse por la compatibilidad con navegadores antiguos.
- Son una herramienta esencial para garantizar que las aplicaciones web funcionen correctamente en diferentes navegadores y versiones, permitiendo a los desarrolladores utilizar las últimas funcionalidades de JavaScript sin sacrificar la compatibilidad.

¿Cómo funcionan los polyfills?

1. Detección: Un polyfill típicamente comienza detectando si la función o API objetivo ya existe en el navegador.

2. Implementación: Si la función no existe, el polyfill proporciona su propia implementación de la función, generalmente usando código JavaScript.

3. Uso: Una vez que el polyfill está cargado, los desarrolladores pueden usar la función como si fuera nativa del navegador.

Ejemplo del proceso de detección de un Polyfill

1. Verificación de existencia: El polyfill verifica si la propiedad o método objetivo está presente en el objeto apropiado. Por ejemplo, para detectar si el navegador soporta la API fetch, se verifica si window.fetch existe.

2. Condicional: Si la propiedad no existe (es decir, es undefined o nula), se considera que la función o API no es soportada y se ejecuta el código del polyfill.

3. Implementación del polyfill: El código del polyfill, generalmente escrito en JavaScript, define la función o método faltante utilizando las características disponibles en el navegador.

4. Reemplazo (opcional): En algunos casos, el polyfill puede reemplazar la implementación original del navegador si esta no cumple con los requisitos o si se necesita una versión mejorada.

Ejemplo del proceso de detección de un Polyfill

1. **Verificación de existencia:** El polyfill verifica si la propiedad o método objetivo

```
if (typeof window.fetch !== 'function') {
  // Polyfill para fetch
  window.fetch = function(url, options) {
    // Implementación del polyfill para fetch
    return new Promise((resolve, reject) => {
      // ... Lógica para realizar una solicitud HTTP usando XHR
    });
  };
}
```

Ejemplo de uso de Polyfill para el fetch

- Inyecta el polyfill fetch al cargar la página, permitiendo que funcione en IE11

```html
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Fetch con polyfill en IE11</title>
</head>
<body>
  <h2>Ejemplo con polyfill de fetch</h2>
  <button onclick="obtenerDatos()">Obtener datos</button>
  <pre id="resultado"></pre>
  <!-- 👇 Polyfill de fetch desde CDN -->
  <script src="https://cdn.jsdelivr.net/npm/whatwg-fetch@3.6.2/dist/fetch.umd.min.js"></script>
  <script>
    function obtenerDatos() {
      fetch('https://jsonplaceholder.typicode.com/posts/1')
        .then(response => response.json())
        .then(data => {
          document.getElementById("resultado").textContent = JSON.stringify(data, null, 2);
        })
        .catch(error => {
          console.error("Error en la solicitud:", error);
        });
    }
  </script>
</body>
</html>
```

Ejemplo de un Polyfill para el map

- Ejemplo de hacer el código de un polyfill para el método map

```
console.log([1,2,3].map(n=>n*2));  // [ 2, 4, 6 ]
// se hace un polyfill para el map usando
// una función callback como sigue:
Array.prototype.customMap = function (callback){
    let result=[];  // arreglo resultante
    for(let index=0; index<this.length;index++){
        const elemento=this[index];
        const elemMultiplicado=callback(elemento,index);
        result.push(elemMultiplicado);
    }
    return result;
};

console.log([1,2,3].customMap(n=>n*2)); // [ 2, 4, 6 ]
```

Beneficios de usar polyfills

- **Compatibilidad entre navegadores:** Permiten usar funcionalidades modernas en navegadores antiguos, evitando errores y comportamientos inesperados. Buscar en **caniuse.com**
- **Desarrollo simplificado:** Los desarrolladores pueden escribir código moderno sin preocuparse por la compatibilidad del navegador, ya que los polyfills se encargan de ello.
- **Actualizaciones graduales:** Los polyfills permiten a los desarrolladores usar las últimas características de JavaScript de manera gradual, en lugar de esperar a que todos los navegadores las admitan nativamente.

Ejemplos que requieren Polyfills

La mayoría de las funciones modernas de JavaScript ya están soportadas por los principales navegadores (Chrome, Edge, Firefox, Safari). Sin embargo hay funciones avanzadas experimentales como:

Función/API	Uso	Estado
`structuredClone()`	Clon profundo de objetos	No soportado en IE, Safari <15
`WeakRef` y `FinalizationRegistry`	Referencias débiles y manejo de GC	Solo en navegadores modernos
`BigInt64Array`	Tipos numéricos grandes	Limitado soporte en Safari <15
Web Streams API (`ReadableStream`, etc.)	Para manejar flujos de datos	No disponible en IE ni versiones viejas de Safari y Android

Funciones modernas de arrays, strings y objetos

Función	Descripción	Navegadores antiguos sin soporte
Array.prototype.at()	Permite acceder con índices negativos	Safari <15, IE, Android antiguo
Object.hasOwn()	Alternativa a hasOwnProperty	No soportado en IE, Safari <16
String.prototype.replaceAll()	Reemplaza todas las coincidencias	IE, Safari <13
Promise.any()	Devuelve la primera promesa que se resuelve	IE, Safari <14.1
Promise.allSettled()	Espera que todas las promesas terminen	IE, Safari <13
Intl.RelativeTimeFormat	Formato como "hace 3 días"	IE, Safari <14
Intl.Segmenter	Divide texto por palabras	Muy limitado (sólo moderno)

Totalmente no soportado (requiere polyfill si se usa)

Función / API	Requiere polyfill completo
fetch() en IE11	✓ Sí (usa whatwg-fetch)
IntersectionObserver en IE/Safari 11	✓ Sí
ResizeObserver en Safari <14	✓ Sí
AbortController en IE y Safari <15	✓ Sí para cancelar fetch()

cedia | TMO Training Management Office

GRACIAS

cedia.edu.ec
info@cedia.org.ec
(+593) 7-407-9300