@VeigaNacho
NachoVeiga

HTML5

WEB WORKERS

Dedicated Workers
Inline Workers
Shared Workers

Características

- Añade la capacidad de procesar scripts en el background de las aplicaciones web.

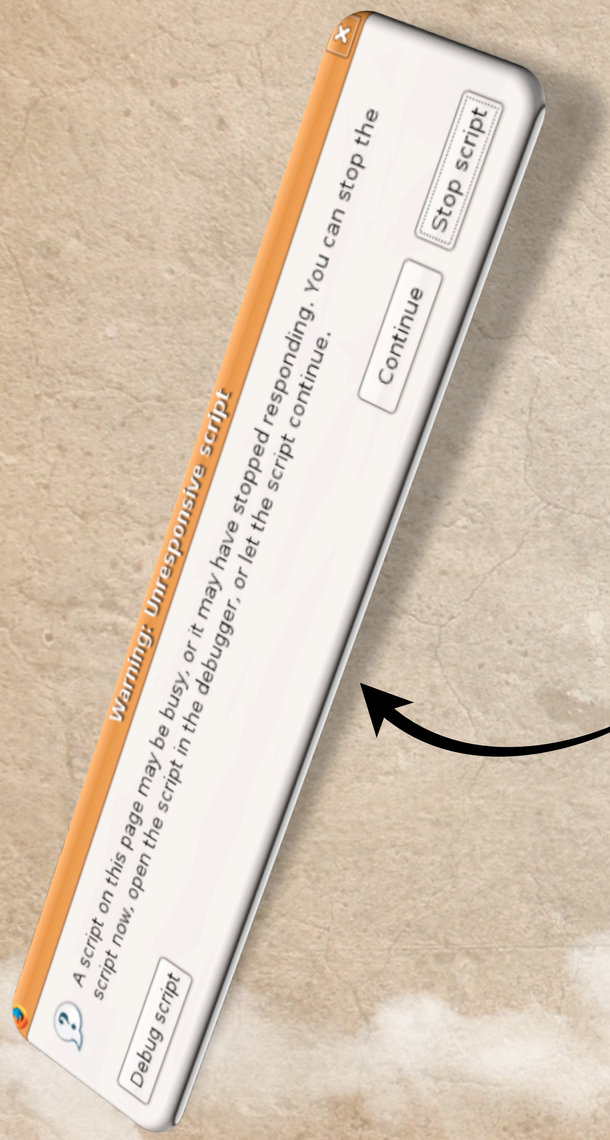
Características

- Añade la capacidad de procesar scripts en el background de las aplicaciones web.
- Cada worker tiene su propio hilo de ejecución.

Características

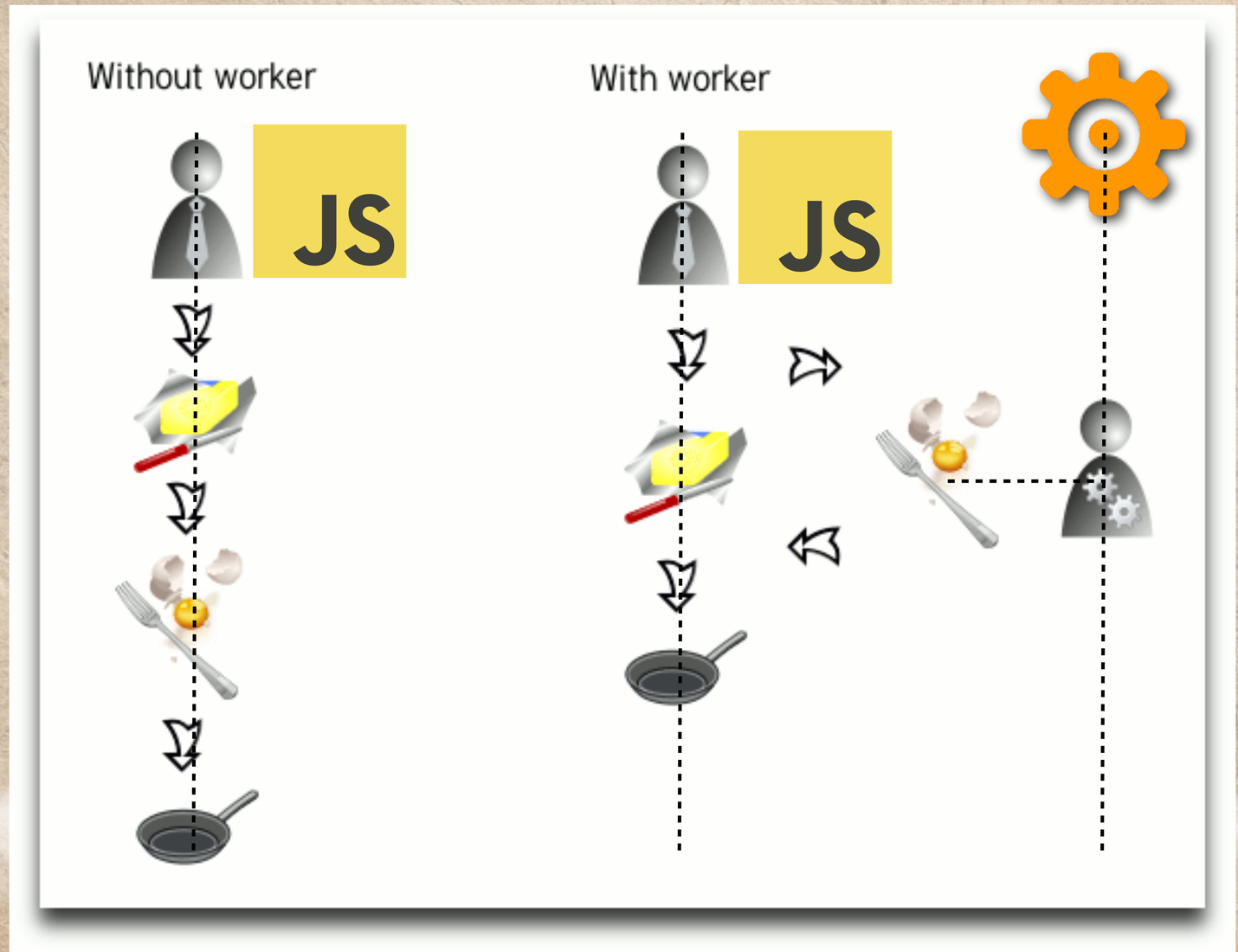
- Añade la capacidad de procesar scripts en el background de las aplicaciones web.
- Cada worker tiene su propio hilo de ejecución.
- Sacan partido de los procesadores multi-núcleo.

Características



- Añade la capacidad de procesar scripts en el background de las aplicaciones web.
- Cada worker tiene su propio hilo de ejecución.
- Sacan partido de los procesadores multi-núcleo.
- Permiten separar tareas pesadas, evitando la aparición de ventanas molestas “Unresponsive script” debido al tiempo de ejecución.

Javascript - Web Workers



Web Workers vs Node + Web Sockets

La ventaja más evidente es la velocidad, los web workers son más rápidos.
Node.js se ejecuta desde un servidor mientras que los workers están en el navegador, lo cual hace, por norma general, que sean más rápidos.
Además, Node necesita su propio servidor lo cual supone un coste extra al proyecto, tanto monetario como de trabajo.

También tienen a su favor que los workers aprovechan el potencial de los procesadores multi-núcleo, como ya comentamos, aunque esto puede ser un problema ya que eventualmente pueden requerir más recursos de los que posees, utilizar toda la ram... utilizad 10 workers como máximo, esto ya sería un poder tremendo.

Un factor a tener en cuenta hoy en día son los dispositivos móviles (tablets y móviles), si utilizáis HTML5 Web Workers + Animaciones CSS3 + JS deberéis probar vuestra aplicación web en diferentes dispositivos para ver si funciona correctamente y fluido.

Limitaciones

Los Web Workers no trabajan en el contexto global de la ventana del navegador, por lo tanto no tienen acceso a:

- El objeto window
- El objeto document
- El objeto console
- El objeto parent
- Y tampoco pueden usar librerías que dependan de estos, adiós Jquery,Prototype...

¿A que tienen acceso?

Debido a su naturaleza multi-hilo, solo tienen acceso a algunas de las características de Javascript, os dejo una lista:

- El objeto navigator
- El objeto location (solo lectura)
- La función XMLHttpRequest (Ajax)
- setTimeout() / clearTimeout() y setInterval() / clearInterval()
- Cache
- Importar scripts usando el método importScripts()
- Web Sockets
- Web SQL Databases
- Web Workers

Compatibilidad

Web Workers - Candidate Recommendation

*Usage stats: Global
Support: 71.74%

Method of running scripts in the background, isolated from the web page

Show all versions	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	IE Mobile
								2.1		
						3.2		2.2		
						4.0-4.1		2.3		
						4.2-4.3		3.0		
	8.0		28.0			5.0-5.1		4.0		
	9.0	23.0	29.0	5.1		6.0-6.1		4.1	7.0	
Current	10.0	24.0	30.0	6.0	16.0	7.0	5.0-7.0	4.2	10.0	10.0
Near future	11.0	25.0	31.0	7.0	17.0					

Sub-features: Shared Web Workers

Notes | Known issues (0) | Resources (5) | Feedback

Edit on GitHub

No notes

Shared Web Workers - Candidate Recommendation

*Usage stats: Global
Support: 41.28%

Method of allowing multiple scripts to communicate with a single web worker.

Show all versions	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	IE Mobile
								2.1		
						3.2		2.2		
						4.0-4.1		2.3		
						4.2-4.3		3.0		
	8.0		28.0			5.0-5.1		4.0		
	9.0	23.0	29.0	5.1		6.0-6.1		4.1	7.0	
Current	10.0	24.0	30.0	6.0	16.0	7.0	5.0-7.0	4.2	10.0	10.0
Near future	11.0	25.0	31.0	7.0	17.0					

Parent feature: Web Workers

Notes | Known issues (0) | Resources (3) | Feedback

Edit on GitHub

No notes

Comprobando la compatibilidad

Existen muchas formas diferentes, os dejo algunas.

- if (typeof(Worker) !== “undefined”)
- if (!window.Worker / window.Worker)
- if (Modernizr.webworkers)

Utilizaré un operador ternario dentro de una IIFE para mostrar un mensaje,después haré una segunda comprobación de la existencia del window.Worker antes de crear los workers.

```
(function(){
var c = "Tu navegador " + ((window.Worker) ? "soporta web workers" : "no soporta web workers");
document.getElementById("compatibilidad").innerHTML = c;
})();
```

Dedicated workers
Crear y detener workers

Creando el worker: var worker = new Worker("workerAlpha.js");

- Utilizamos la función Worker(URL).
- La URL corresponde con el fichero JavaScript con el código del worker.
- Se asigna a una variable, en este caso worker.

¿Como lo detengo?

- Desde el propio worker : self.close();
- Desde fuera del worker : worker.terminate();

Comunicación con workers
Enviando mensajes

Los web workers, se comunican mediante el paso de mensajes, utilizando el evento “MessageEvent".

El “data” o información enviada entre la página principal y el worker es copiada, no compartida.

Los objetos son primeramente serializados, y, posteriormente, de-serializados en el otro lado (pagina principal/ worker) .

La página principal y el worker no comparten la misma instancia, el resultado es que se crea un duplicado en cada una de ellas, la mayoría de los navegadores implementan esta característica como “structured clonning”.

Comunicación con workers “structured clone” y serialización

El algoritmo "structured clone" es un nuevo algoritmo definido por la especificación de HTML 5 para serializar los objetos JavaScript complejos.

La serialización es el proceso de convertir el estado de un objeto en un formato que se pueda almacenar o transportar.

El complemento de serialización es deserialización, que convierte una secuencia en un objeto.

En este caso se utiliza para crear un nuevo objeto que es idéntico en todo al original, es un clon.

Comunicación con workers
Enviando mensajes – Ejemplos

Enviamos los mensajes con la función postMessage() haciendo referencia al worker o utilizando self desde el propio worker.

Podemos enviar strings,números, valores booleanos, arrays, objetos javascript y json.

Ejemplos:

- worker.postMessage('Hola Mundo');
- self.postMessage(506);
- function enviarMensaje(me){
  worker.postMessage(me)
  ;}
- worker.postMessage(personajes)

- var personajes = [
{Nombre: "Chuck" , Apellido: 64},
{Nombre: true , Apellido: "Grimes"},
{Nombre: "Walter" , Apellido: "White"}
];

Comunicación con workers
Recibiendo mensajes

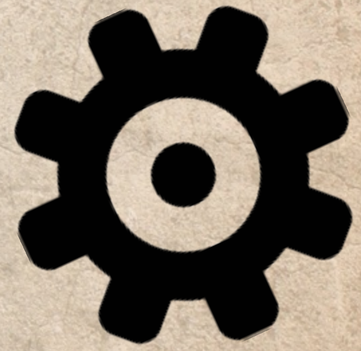
Tenemos que escuchar el evento “message”, tanto en el worker como el en hilo principal.

<u>Podemos utilizar un Listener:</u>

- worker.addEventListener(“message”, function(){});
- self. addEventListener(“message”, function(){});

<u>o directamente -> worker.onmessage:</u>

- worker.onmessage = function (){};
- self.onmessage = function (){};



MessageEvent – data y timeStamp

Dentro del evento MessageEvent, nos interesan sobre todo el data y el timeStamp:

•data: contiene el información enviada.
•timeStamp: contiene la fecha y hora, por desgracia en un formato no legible, nos muestra los milisegundos desde 1/1/1970 -> 1381272453343

Simplemente creamos un nuevo date, utilizando el timeStamp, y ya podremos mostrar la información de una forma más legible, ademas podremos usar los métodos del objeto Date: getDay, getHours,getMonth,getSeconds,getFullYear...

```
var fechaYhora =  new Date(me.timeStamp);
```

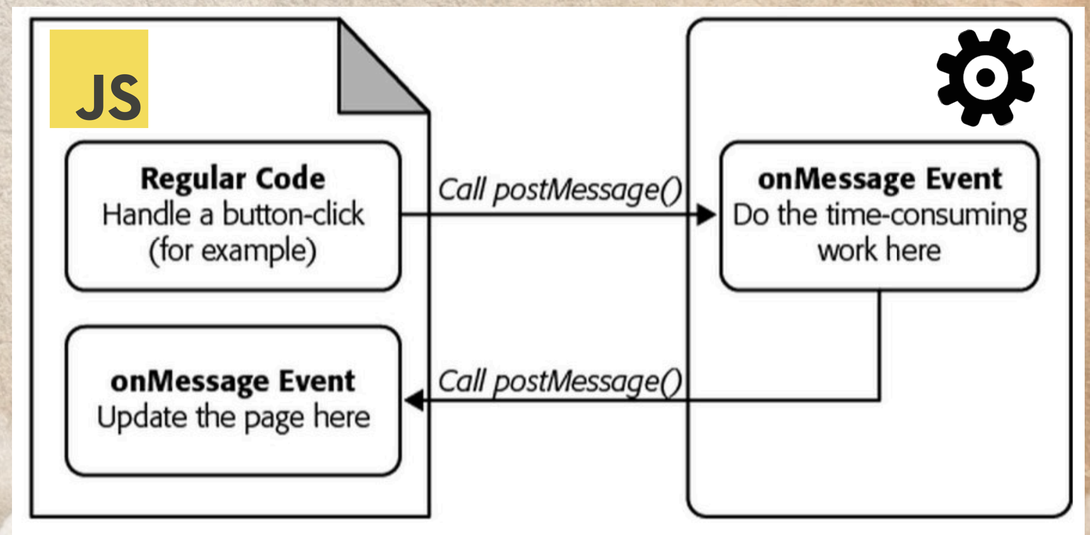
JSON – stringify y parse

Los Workers tienen soporte para JSON y por lo tanto también a sus métodos stringify() y parse().

Podéis usar el método parse() para ahorrar tiempo al “parsear” objetos JSON string largos. Un worker simple para hacer esto sería:

-- > parserWorker.js < --

```
self.onmessage = function(me){
  var jsonString = me.data;
  var jsonObject = JSON.parse(jsonString);
  self.postMessage(jsonObject);
};
```



Import Scripts

Los workers tienen acceso a la función global importScripts(), la cual te permite importar scripts o librerías dentro de su scope, acepta como parámetros cero o más URIs

```
importScripts();
importScripts('script.js');
importScripts('libreria.js', 'cdn.js');
```

Los scrips se ejecutan en el orden que tu estableces y se hace de forma síncrona, importScripts() no regresa hasta que todos los scripts se hayan cargado y ejecutado.

Inline Workers – 1/2

Vamos a crearlos utilizando el objeto Blob(), existe otro método mediante el BlobBuilder pero quedo obsoleto en favor del objeto Blob, la implementación con este último es mucho más clara.

Para crear un inline worker necesitas 3 objetos: el worker, el Blob y el Url.

1º- Primero configuramos el inline worker, estableciendo un id para poder acceder a él más tarde, le ponemos type=”app/worker”, este script será el equivalente a un archivo independiente de un worker

```
<script id="worker" type="app/worker">
    self.onmessage =function(me) {
        postMessage('Data: ' + me.data );
    };
</script>
```

Inline Workers – 2/2

2º - Creamos el blob (burbuja)

Creamos la burbuja usando el objeto Blob() y le pasamos el contenido del script que acabamos de crear, debe de estar en un array:

```
var blob = new Blob([document.getElementById('worker').textContent]);
```

3º - Creamos la url

Creamos la referencia al script del worker.

```
var url = window.URL.createObjectURL(blob);
```

4º - Instanciamos el Worker

Por último, instanciamos un nuevo worker.

Al worker le pasamos la url como argumento.

```
var worker = new Worker(url);
```

Patrón Inline Worker

```
<body>
<script id="worker" type="app/worker">
        self.onmessage =function(me) {
        postMessage(me);
      };
</script>
<script>
   function inlineWorker() {
      var blob = new Blob([document.getElementById(" worker ").textContent]),
            url    = window.URL.createObjectURL(blob),
            wor  = new Worker(url);

      wor.onmessage =function(me) {
      console.log();
      };
    wor.postMessage(" ");
    };
    window.onload = inlineWorker();
  </script>
</body>
```

Shared Workers 1/3

Los shared workers permiten que varios scrips, incluso de diferentes páginas, se conecten de forma simultánea al mismo sharedworker, pero deben respetar el “same origin policy” es decir, mismo protocolo, mismo host y mismo puerto.

Los shared workers ademas de lo ya mencionado en los anteriores tipos de workers, tienen el atributo “port”, el evento “connect” y el método start()

1º- Primero vamos a crear el script del worker y una variable para controlar a los clientes que se van conectando

var cliente;

Shared Workers 2/3

2º Creamos un listener para el evento connect

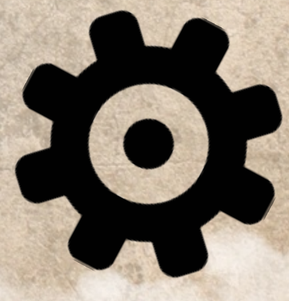
3º Dentro del listener del connect, vamos a guardar el puerto en una variable

4º Creamos un listener para el evento message utilizando la variable port

5º Ahora creamos el port.postMessage y el port.start();

```
self.addEventListener("connect", function (e) {
    var port = e.ports[0];
    cliente++;

    port.addEventListener("message", function (me) {
        port.postMessage("Puerto #" + cliente);
    });
    port.start();
});
```



Shared Workers 3/3

6º- Finalmente creamos el SharedWorker.

var worker = new SharedWorker("sharedworker.js");

El resto es similar a los casos anteriores, pero recordad agregar el “port” en el addEventListener(), start(), y postMessage().

```
worker.port.addEventListener("message", function(me) {
  console.log(me.data);
});

worker.port.start();

worker.port.postMessage("");
```

Patrón SharedWorker.js

```
var cliente;

self.addEventListener("connect", function (e) {

    var port = e.ports[0];

    cliente++;

    port.addEventListener("message", function (me) {
      port.postMessage("Puerto #" + cliente);
    });

    port.start();

}, );
```

Patrón SharedWorker.html

```
<script>

var worker = new SharedWorker("sharedWorker.js");

worker.port.addEventListener("message", function(me) {
  console.log(me.data);
});

worker.port.start();

worker.port.postMessage("");

</script>
```

Gestión de errores

Para lidiar con los errores, tenemos el evento “error”, para mirar más detalladamente los errores, podemos usar:

- filename - Nombre del Archivo
- lineno - La linea en donde ocurrió el error
- message - Descripción del error

Gestion de errores

```
worker.onerror = function workerError(error) {
    console.log(error);
}
worker.addEventListener('error', function(error){
    console.log(' Error en archivo: '+error.filename
        + '\n Linea: '+error.lineno
        + '\n Mensaje: '+error.message);
});

// Sin tener x definida
self.postMessage(x);
```

Gracias por vuestra atención

Ejemplos de web workers:

HTTPS://DEVELOPER.MOZILLA.ORG/MS/DEMOS/TAG/TECH:WEBWORKERS