**WEB WORKERS – HTML5**

* HTML5’s new API
* Allows us to execute some JS in background of page without interrupting the current running JS and page performance
* Mimic multithreading as JS runs in background without affecting user interface

**WHAT IS IT?**

* Browser runs web workers on different thread than UI thus provides multi-threading behavior
* A worker is an object created using a constructor (e.g. [Worker()](https://developer.mozilla.org/en-US/docs/Web/API/Worker/Worker)) that runs a named JavaScript file
* this file contains the code that will run in the worker thread
* workers run in another global context that is different from the current [window](https://developer.mozilla.org/en-US/docs/Web/API/Window)
* worker object is accessible through self keyword

For e.g. :

<!DOCTYPE HTML>

<html>

<head>

<title>Worker example: One-core computation</title>

</head>

<body>

<p>The highest prime number discovered so far is: <output id="result"></output></p>

<script>

if(window.Worker) {

var worker = new Worker('worker.js');

worker.onmessage = function (event) {

document.getElementById('result').textContent = event.data;

};

}else {

console.log(“browser does not support workers”);

}

</script>

</body>

</html>

**Worker.js**

var n = 1;

search: while (true) {

n += 1;

for (var i = 2; i <= Math.sqrt(n); i += 1)

if (n % i == 0)

continue search;

// found a prime!

self.postMessage(n);

}

**WHAT WE EXPECT FROM WEB-WORKER API:**

* workers are expected to be long-lived, have a high start-up performance cost, and a high per-instance memory cost.
* Not intended to be used in large numbers. For example, it would be inappropriate to launch one worker for each pixel of a four megapixel image.

**COMMUNICATION BETWEEN MAIN THREAD AND WORKERS :-**

* communication is done using message event and event handler (e.g. onmessage and postMessage)
* both side can share and respond to messages
* message which is passed can be either string or object

|  |  |
| --- | --- |
| Main thread (main.js) | Worker thread(worker.js) |
| var worker = new Worker(worker.js);  worker.postMessage(“main thread”);  worker.addEventListener(“message”,function(evt){  console.log(evt.data);  }, false); | self.addEventListener(“message”, function(evt){  console.log(evt.data);  self.postMessage(evt.data + “ worker thread”);  },false); |
| Output : main thread worker thread | Output : main thread |

**Scopes :**

* The global scope is the "inside" of a worker.
* The **self** attribute must return the WorkerGlobalScope object itself.

**Types of Worker :**

* Dedicated Web Workers
* Shared Web Workers

|  |  |  |
| --- | --- | --- |
|  | Dedicated Web Worker | Shared Web Worker |
|  | * Standard Worker utilized by a single Script( Linked to creator) | * Allow any script of same domain/origin to obtain the reference of worker and communicate with it |
| Initialization | * var dWorker = new worker(“worker.js”); | * var sWorker = new SharedWorker(“shared\_worker.js”); |
| Listening & passing events | * dWorker.addEventListener(“message”, eventHandler, false); * dWorker.postMessage(“msg”); | * sWorker.port.addEventListener(“message”, eventHandler, false); * sWorker.port.postMessage(“msg”); * sWorker.port.start(); |
| Closing a worker | * dWorker.terminate(); | * sWorker.terminate(); |
|  | * each worker have only one connection | * one Worker can have multiple connection |
|  |  |  |

**What we can not access in Web-Workers:**

Since web workers are in external files, they do not have access to the following JavaScript objects:

* Window Object
* DOM Object
* Parent Object

Main App Memory/Object

**What we can access in Web-Workers:**

* XHR Requests
* Navigator,location
* AppCache
* ImportScript

**Browser’s Support**:

* Chrome 4.0+
* Firefox 3.5+
* Safari 4.0+
* Opera 11.5+
* iOS Safari 5.1+
* IE 10+