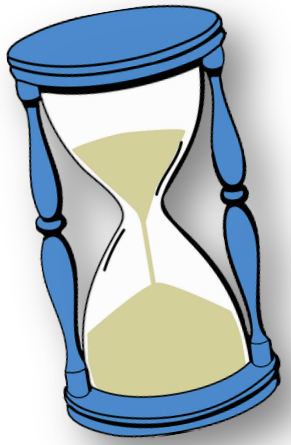
WEB WORKERS

By: Louis BOUDDHOU

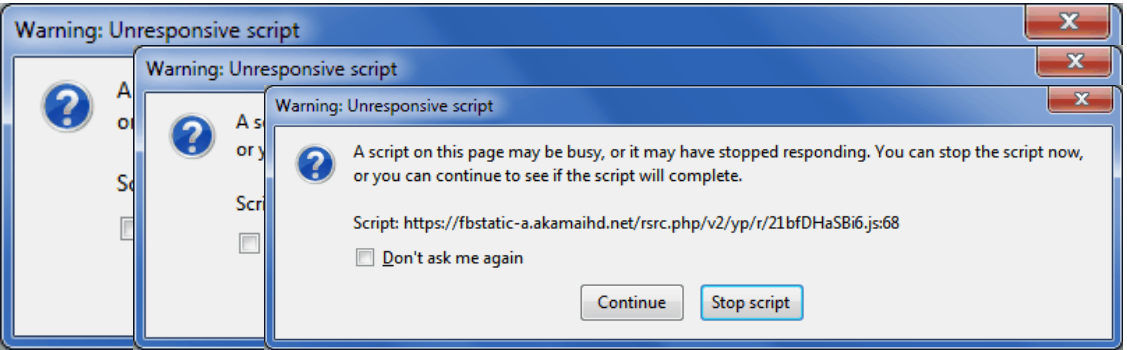
WEB WORKERS

PROBLEM:

- JavaScript is a single-threaded environment.
- Multiple scripts cannot run at the same time.
- Imagine a site that needs to handle UI events, query and process large amounts of API data, and manipulate the DOM.
    - They can't all be simultaneous due to limitations in browsers' JavaScript runtime.
    - Script execution happens within a single thread.



WEB WORKERS



"We all know the bad consequences of overloading [a] thread: the page freezes and the user can't interact with [the] application any more."

"The user experience is very unpleasant, and the user will probably decide to kill the tab or the browser instance."

WEB WORKERS

The JavaScript Multi-threaded Approach

- Web Workers lets you write true multi-threaded JavaScript
- It provides a facility for creating new threads for executing your JavaScript code in.
- A multi-threaded architecture is created in which the browser can execute multiple tasks at once. Creating new threads for handling large tasks allows you to ensure that your app stays responsive and doesn't freeze up.

WEB WORKERS

A Simple Code Example

- Using a web worker requires 2 separate JavaScript files.
- One is executed in the UI thread. We'll name it ***main.js***:

```
1  worker = new Worker('worker.js');
2  worker.addEventListener('message', receiveMessage);
3
4  function receiveMessage(e) {
5      console.log(e.data);
6  }
7
8  worker.postMessage('cowboy');
```

- This example assumes the 2 files are sitting side by side.

WEB WORKERS

- And the other (we'll name it ***worker.js***) will run in a background thread:

```
1  self.addEventListener('message', receiveMessage);
2
3  function receiveMessage(e) {
4      self.postMessage('Sup, ' + e.data + '?');
5  }
```

- ***main.js*** instantiates the worker object with the URL for ***worker.js***
- Both files set up their own message listeners, each with a callback to their own `receive Message` function.
- This is how the UI thread and background thread communicate: via messaging. They both listen for messages from each other and can post messages to each other.

WEB WORKERS

- The final line of ***main.js*** is what kicks things off. Here's the flow:
  1. ***main.js*** messages 'cowboy' to ***worker.js***
  2. ***worker.js*** receives the message, handles it by doing a little string concatenation, then messages 'Sup, cowboy?' back to ***main.js***
  3. ***main.js*** receives this message and handles it by logging 'Sup, cowboy?' to the console

Richer Message

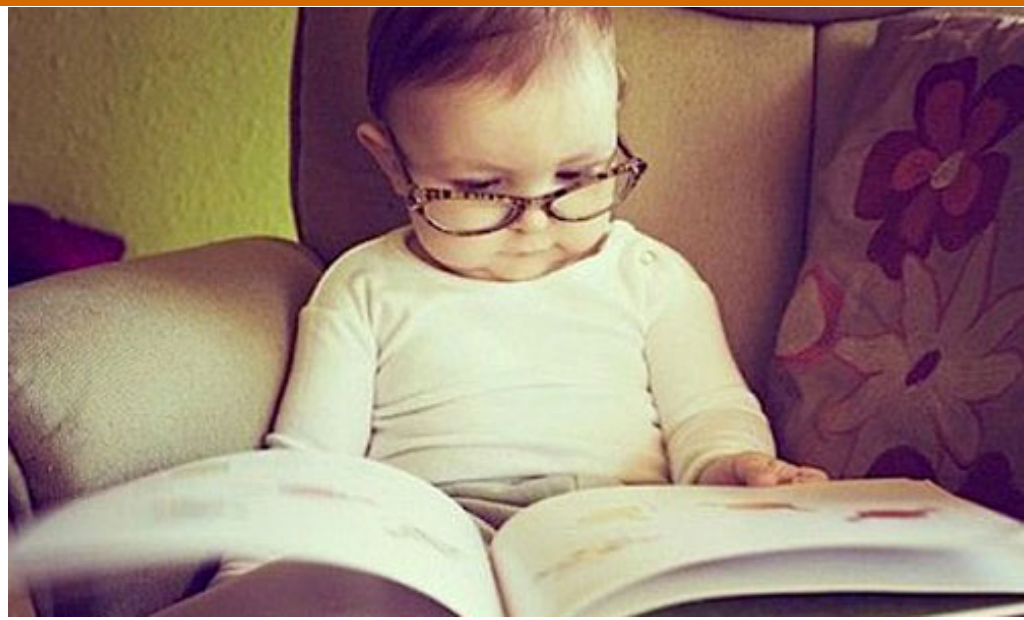
- The sample code above uses very simple data (just a string) for each message, but you can send more complex data like so:

```
worker.postMessage({
    operation: 'update',
    id: 41,
    occupation: 'cowboy',
    date: new Date()
});
```

- In this case, the web worker could access the occupation value via e.data.occupation. Yeah, it's JSON.
- You can't send functions via messaging.

WEB WORKERS

Application



- Web workers must be put into their own file.
- Their code lives in a completely separate context than any code that exists in the UI thread.
- Only messaging can cross the divide.
- There are no shared global variables.
- Even the data sent via `postMessage` is not shared; it's cloned.
- Web workers do not use `window` (they use `self` instead).

WEB WORKERS

However:

- Web workers cannot access the DOM (territory of the UI thread).
- It should post a message to the UI thread telling it to make any DOM updates.
- These restrictions make things much simpler. Multi-threading headaches found in other languages (like locking objects) don't apply to JavaScript, thanks to its philosophy of separate contexts and not sharing objects.

WEB WORKERS

Browser Support

IE	Firefox	Chrome	Safari	Opera	iOS Safari	Android Browser	Chrome for Android
8		27				2.3	
9		31				4	
10	28	32	5.1			4.1	
11	29	33	6.1			4.3	
	30	34	7	21	6.1	4.4	35
	31	35		22	7.0		
	32	36		23			
		37					
		38					

WEB WORKERS

Sources

- http://www.htmlgoodies.com/html5/tutorials/introducing-html-5-web-workers-bringing-multi-threading-to-javascript.html#fbid=4aUfB5wnFVG
- http://blog.teamtreehouse.com/using-web-workers-to-speed-up-your-javascript-applications
- http://codersblock.com/blog/multi-threaded-javascript-with-web-workers/
- https://msdn.microsoft.com/en-us/hh549259.aspx
- https://www.youtube.com/watch?v=rip-txJX65g