Web Workers

CS-554 – WEB PROGRAMMING

What are Web Workers?

Web workers? Is that us?

While we work on the web, Web Workers is a term for two types of objects that can be created from your browser:

- Dedicated Workers
- Shared Workers

These enable us to run some amount of JavaScript in a “different thread” than our main applications.

What do workers have access to?

Workers do **not** have DOM and browser information related APIs. Workers commonly utilize:

- XMLHttpRequest
- IndexedDB
- Websockets
- Fetch (Start trying to make Fetch happen…)
- WebGL & Offscreen Canvas (you can draw on a worker and send back the result to the client! Oh my!)

In addition, Workers have a few Worker specific things:

- importScripts; the ability to download and execute a number of scripts in the worker's context (basically, "add a script tag in a worker")
- postMessage; the ability to post a message back to the caller (or another port)
- onmessage: a function to be run when a message is received

Dedicated Workers

Dedicated Workers are ”one-page” workers; their scope is isolated to the page that spawned them. The worker’s scope is **not** shared with other pages, iframes, etc.

Dedicated workers can be viewed as the personal worker thread for a web page.

Example Dedicated Worker

For our example, we will have our worker search a familiar data set:

- https://gist.github.com/philbarresi/5cf15393d245b38a2d86ce8207d5076c

We will have our main thread submit a request, and all entries will be searched for the substring submitted from our main thread.

Example

A brief example has been written using **Flow** and **Typescript**.

This example has a page and 2 iFrames. The frames and the main page all point to **sharedWorker.js** (built with flow!).

- Our main page will allow users to send a message.
- They will all communicate through a SharedWorker pointing at sharedWorker.js
- When a user sends a message, it will appear on every page using that shared worker

Shared Workers

Shared Workers are ”many-page” workers; you can load a particular worker script up on several pages, iframes, tabs, etc. Scope will be shared across each instance of those workers, allowing you to pass data to many locations.

- Sometimes, however, opening up a new tab does not share the scope. Browsers are weird ¯_(ツ)_/¯

Shared Workers are incredibly useful for maintaining a single socket connection for a domain, and passing messages off to each port on the worker; this allows one worker to talk to the server, and then talk to each page rather than having to burn many resources on multiple connections.

Service Workers

Service Workers are a more advanced Worker that can do things like act as proxy servers and allow for offline applications to be developed with a great degree of control.

We will look at Service Workers during our final lecture.

Sharing Data With Workers

The easiest way to share data with Workers is to use **worker.postMessage(data)** to send data to the Worker.

- This performers a structured clone of the data; it basically can be seen as creating a “naïve clone” of the object and sending it to the worker

Performing a structured clone is expensive. There is another way to transfer certain types of objects; objects which inherit from the Transferable interface can be moved from the one context to another (i.e., a main script to a worker).

- worker.postMessage({foo: transferListObject1, bar: transferListObject2}, [transferListObject1, transferListObject2])

Transferable objects are currently ArrayBuffer, MessagePort, and ImageBitmap objects.

If an object is transferred, it is unusable in the context from which it was transferred until the object is transferred back to the sender.