WEB WORKERS

by Maksym Khudoliy

softserve

WEB WORKERS

Web Workers allow you to perform **long in time** and **complex** tasks **without blocking** the user interface, i.e. performing complex work, the browser will continue to respond quickly to user actions. Web Workers run on a **separate**, **isolated thread**, so the Web Workers code must be kept in a separate file. They do **not have access** to the "**window**" and "**document**" objects, i.e. Web Workers cannot directly manipulate user interface graphics

Examples of tasks where you should consider using Web Workers:

- preloading data
- large data caching
- analysis of text/audio/video according to various criteria
- encryption

WEB WORKERS

To start working with Web Workers, you need to create **Worker** object type, the parameter of which specifies the **name of the file** with the code. If the specified file does **not exist**, there will be a **404** error and the object will not be created

The **interaction** between the **main thread** of the Web page and the **Web Workers** is based on the **event system** and **message** passing:

- to transfer data, the **postMessage(data)** method is used, the parameter of which is the data to be transferred
- to receive data, the "**message**" event is used, into the handler of which an object of the **MessageEvent** type will be passed, the "**data**" property of which contains the transferred data

WEB WORKERS

Please note, that the postMessage() method and the "message" event are **used by both** the main thread of the Web page and Web Workers, i.e. if the main thread of the Web page wants to transfer data to Web Workers, it calls postMessage(), the same does the Web Workers if it wants to transfer data to the main thread of the Web page

There are two ways to stop Web Workers:

- call the **terminate()** method on the main thread of the web page
- call **close()** method inside Web Workers

WEB WORKERS

To demonstrate how Web Workers work, let's create the following files:

- **index.js** – the code of the main thread of the Web page
- **worker.js** – Web Workers code
- **index.html** – Web page, only needed to include index.js

Please note, that the files must be placed on **any available server**. If you run the code without a server, directly in the browser, Web Workers will not work

WEB WORKERS

index.js file:

```
const w = new Worker("worker.js");
w.addEventListener("message", (e) => {
   console.log("Main <- Worker:", e.data);
});
w.postMessage("Hello Worker!");
setTimeout(() => w.postMessage("Hello World!"), 1000);
setTimeout(() => w.postMessage("stop"), 2000);
```

We create Worker object type. Add a handler to the "message" event to receive messages from Web Workers. We send data three times with a small delay

WEB WORKERS

worker.js file:

```
console.log("Worker started");
addEventListener("message", (e) => {
   if (e.data !== "stop") {
      console.log("Main -> Worker:", e.data);
      postMessage("data received");
   } else {
      close();
      console.log("Worker stopped");
   }
});
```

softserve

WEB WORKERS

We display the start of work in the browser console. Add a handler to the "message" event to receive messages from the main thread of the Web page. If the main thread of the Web page passes the string "stop", we stop working

After running the code, the browser console will display the following output:

```
Worker started

Main -> Worker: Hello Worker!

Main <- Worker: data received

Main -> Worker: Hello World!

Main <- Worker: data received

Worker stopped
```

softserve

WEB WORKERS

During the execution of Web Workers, **errors** may occur, the "**error**" event is responsible for handling errors, and an error object and its useful properties will be passed to its handler:

- **filename** – the name of the file that contains the script that caused the error
- **lineno** – the line number where the error occurred
- **message** – error description

softserve

WEB WORKERS

An example of error handling. **index.js** file:

```
const w = new Worker("worker.js");
w.addEventListener("error", (e) => {
   console.log("File:", e.filename);
   console.log("Error in", e.lineno, "line");
   console.log("Message:", e.message);
});
w.postMessage("Hello Worker!");
```

WEB WORKERS

worker.js file:

```
console.log("Worker started");
addEventListener("message", (e) => {
   console.log(x);
});
```

After running the code, the browser console will display the following output:

```
Worker started
File: http://localhost:3000/worker.js
Error in 5 line
Message: Uncaught ReferenceError: x is not defined
```

softserve

TO
THE
FUTURE
softserve