Sofascore Frontend Academy Lecture 04,
March, 2023

Advanced JavaScript

Alen Murtić

Sofascore

TABLE OF CONTENTS

01

JavaScript runtime

JavaScript runtime

JavaScript Design

- Single-threaded -> no threading and standard parallelization for developers
  - Has only one thread, which is often called *Main thread*
- Concurrent
  - Based on *event loop*
  - one operation can progress without waiting for another operation to finish if the thread is available
  - (e.g. JS will not wait for an HTTP request to finish and JS will run next line of a script).
- Event loop -> collects, processes, executes sub-tasks (asynchronous code) and makes concurrency possible.

JavaScript Runtime

- Semantics that JavaScript engines implement and optimize
  - parts that should be implemented by JavaScript engines (e.g. V8)
- Defines how JavaScript should run
- Contains heap and call stack
  - as in other languages, a heap is used for memory allocation (e.g. this is where variables are stored, etc.)
  - stack stores information about active functions (it keeps track of the script flow)
- Call stack answers questions:
  - where is the script currently (which function)
  - where should the script return after the current function
  - what variables are currently in the scope

JavaScript runtime

Blocking

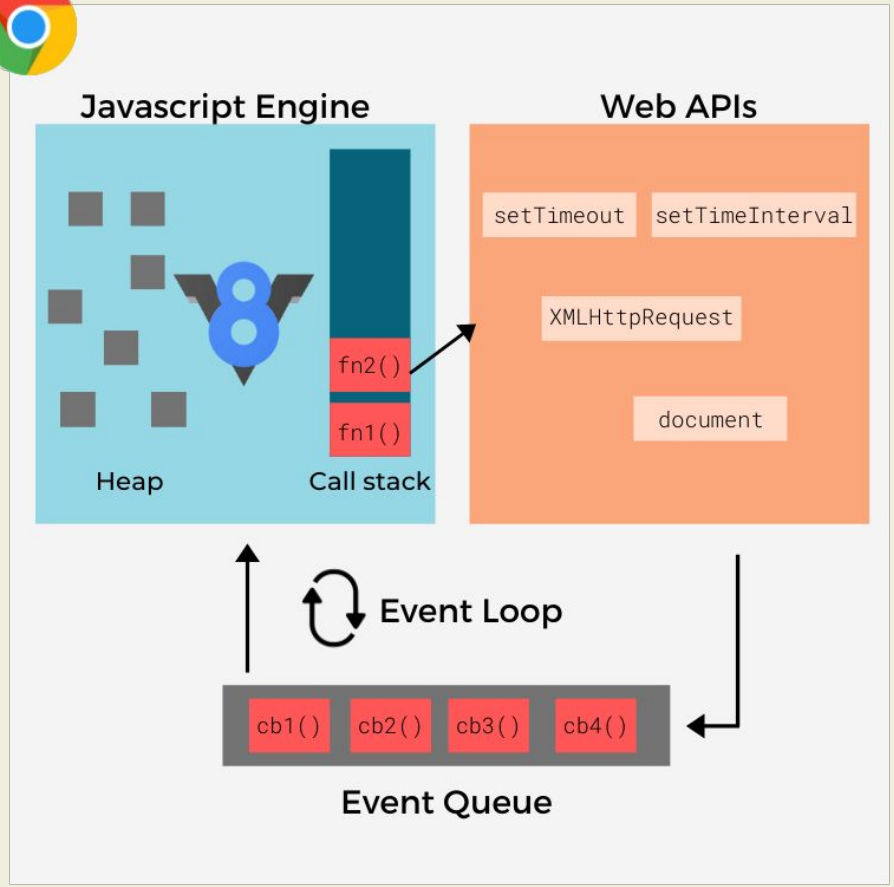
- Happens when some function on the call stack(main thread) is taking too long (> 200ms) to execute so others can't execute
- Direct consequence of the single-threaded runtime design
  - Only one thing can run at a time on a call stack
  - Others are blocked
- JavaScript handles blocking and non-blocking functions
- Developers should avoid blocking synchronous code whenever possible
  - BUT: Writing 100% non-blocking code is impossible
- Non-blocking code usually executes in "background" (it uses other browser resources - networking, I/O operations, ...) and calls the function with the result once it has finished
  - These functions are called **callbacks**.

Blocking code example - blocking.html, nonBlocking.js

JavaScript runtime

Browser overview

- Runtime -> Runs code, single-threaded
- WebAPIs -> Functions called by our JS scripts, implemented by browsers or platforms (Node.js)
- Callback queue -> Queues function that should be executed when possible. Those functions come from WebAPIs

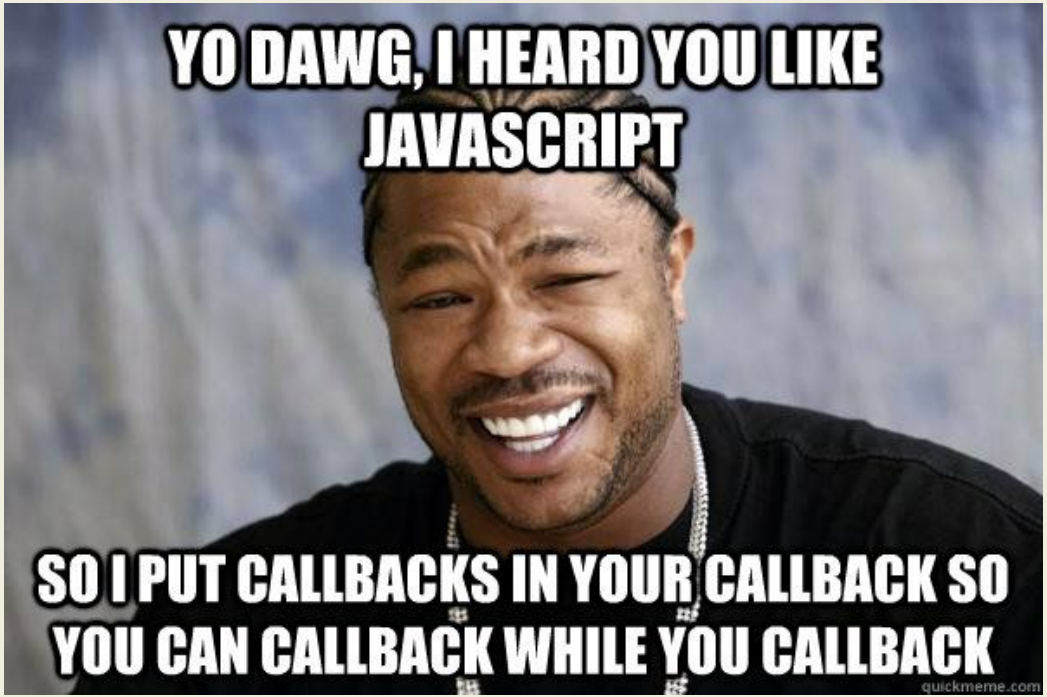


JavaScript runtime

Callback hell

- A function depends on another function which depends on yet another function ...

```
doAsyncWork(() => {
  // work done, do another work
  doAsyncWork(() => {
    // work done, do another work
    doAsyncWork(() => {
      // work done, do another work
      doAsyncWork(() => {
        // WELCOME TO CALLBACK HELL
      })
    })
  })
})
```



02

Promises

Promises

- Object that represents the eventual completion of an asynchronous operation
- Has 3 states:
  - Pending -> Executing, not completed yet
  - Fulfilled -> Completed successfully
  - Rejected -> Completed, but not successfully (error happened)
- Too much information (but it's good documentation, I *promise*)
- Everything can be wrapped in the promise

Promises


PROMISES ARE EITHER
Kept
or Broken
Resolved
or Rejected
imgflip.com

Promises example - promises.js

Promises

Async-await

- Developer-friendly way to handle promises
- Asynchronous functions can be declared with the async keyword
  - async functions always return promises
- await keyword pauses the execution of the async function until the promise is fulfilled
  - Can be only used inside the async declared function
  - **Doesn't block non-related code execution**
  - Should be wrapped in a try-catch block
    - If the promise is resolved successfully, its data will be stored into a provided variable, and execution will continue
    - If the promise is rejected, try-catch block will catch the error

Async-await example - asyncAwait.js

03

Event Propagation

Event Propagation

Event Handlers

- HTML DOM allows JavaScript to register Event handlers on elements
- Part of WebAPIs
- Handler executions are put onto the callback queue
  - click, change, focus, blur, load, scroll, ...
  - Full list of events
  - Use scroll sparingly -> triggers on every scroll frame -> can flood the callback queue
    - Or: debounce it - caring about state every e.g. 50 ms instead of on each scroll

Event Propagation

- What happens when the event occurs in the DOM
- Which event handler(s) should be called and in which order
- Two modes:
  - Event Bubbling - BOTTOM -> TOP
    - Event is handled from the nearest listener (on the element where the event occurred or parent). Propagation continues to the next parent element and so on
  - Event Capturing - TOP -> BOTTOM, also called trickle mode
    - Event is handled from the first listener in the DOM (from the top), then propagation continues to the bottom of the DOM.

Event Propagation example - eventPropagation.html

04

Fetching data

Fetching data

- Part of the WebAPIs
  - fetch function (from window)
- Fetch has promise based API -> returns promise
- Docs on API with usage examples, docs on fetch function

Data fetching example - fetch.html

05

Typescript

Before Typescript

- JavaScript is dynamically typed and has "quirky" behaviors
  - These and other features/shortcomings make it easier to make mistakes in JS
- Should JS be replaced as THE language of the web?
  - Bigger undertaking than writing in another language and compiling the code into JS
  - 2009-2010: CoffeeScript which compiles into JS
  - 2011: Google Dart which compiles into JS

Typescript

- 2012: Microsoft announces TypeScript (TS), a superset of JavaScript
  - Meaning every JS code is valid TS code, but you can write better code
  - With future ES specifications in mind
  - Gradually released, but became a big hit!
  - Complies .ts files into regular .js and .tsx into .jsx (React components are written in .jsx files)

Typescript

TypeScript syntax

- Syntax
  - Four essential cheat sheets
  - The gist of TS are TYPES, i.e. only describing values of variables, object properties, etc.
  - Everyday types
  - TS does add enums as a feature
- Typescript + React
  - Great combination, React is much more expressive with TypeScript
  - Types are added as extra modules to existing libraries, they don't have to be re-written
  - Your TS code is transpiled into desired version of JS code

Typescript example - typescriptModel.ts

Thank you for your attention!

Sofascore