Асинхронность на уровне движка JS. Цикл событий.

Для того, чтобы были возможны асинхронные операции в языке JS, в движке существует механизм, называемый циклом событий (Event loop) [1]. Он беспрерывно обслуживает следующие структуры, также реализованные в движке JS:

- Стек вызовов;
- Очередь макрозадач;
- Очередь микрозадач.

Стек вызовов (Call stack) – структура данных, реализованная в движке JS для отслеживания текущей позиции выполнения скрипта на уровне JS-функций. Он подчиняется принципу организации данных LIFO (Last In – First Out). Вызов каждой функции помещается в стек вызовов в виде стекового фрейма функции, содержащего все её локальные переменные и параметры. После отработки функции (и всех ее вложенных функций), запись о её вызове извлекается из стека и управление передается в точку возврата, которая была передана (в виде адреса памяти) в стек при вызове функции [1, 2, 3].

Очередь макрозадач (Macrotask queue), также называемая часто очередью обратных вызовов (Callback queue) – еще одна структура данных, также реализованная в движке JS. Она подчиняется принципу организации данных FIFO (First In – First Out) и состоит из задач, каждая из которых связана с определенной функцией – как правило это коллбэк, переданный в некоторые асинхронные функции, такие как fetch, setTimeout и т. д. Занесение задачи в очередь происходит в момент вызова асинхронной функции [4].

Очередь микрозадач (Microtask queue) аналогична очереди макрозадач, однако задачи в ней ассоциированы с коллбэками, передаваемыми в .then(), .catch() и .finally() в промисы (Promises), и с блоком инструкций асинхронной функции, следующими сразу после await и до конца функции [4].

Получается, что цикл событий обслуживает в первую очередь стек вызовов (Call stack), то есть выполняет синхронный код. Как только этот стек опустошится (кроме главной функции – точки входа в программу), цикл событий проверяет наличие микрозадач в очереди (Microtask

queue), и, если там есть хоть одна, то в стек вызовов будут сразу занесены стековые фреймы как самого коллбэка, ассоциированного с первой задачей в очереди, так и всех вложенных в него вызовов функций. Если же очередь микрозадач пуста, то аналогично обслуживается очередь макрозадач (Macrotask queue). То есть, очередь микрозадач имеет приоритет перед очередью макрозадач. После очередного опустошения стека вызовов весь процесс циклически повторяется.

Подробное объяснение механизма цикла событий вы можете найти в статье [4]. Также настоятельно рекомендуется уделить время просмотру замечательного видео [5] об Event Loop, записанного с международной конференции JSConf EU 2014.

Вопросы и задания

1. Можно ли гарантированно утверждать, что вызов setTimeout(cb, 0) гарантированно выполнит cb (некий переданный коллбэк) через 0 секунд? Объясните почему.

2. Изобразите в виде блок-схемы или графа процесс работы цикла событий.

3. Реализуйте средствами языка JS структуру данных Stack (стек) на массиве, подчиняющуюся принципу LIFO, с методами push() и pop().

4. Реализуйте средствами языка JS структуру данных Queue (очередь) на массиве и связном списке, подчиняющуюся принципу FIFO, с методами enqueue() и dequeue().

5. Опциональное задание. Реализуйте средствами языка JS структуру данных Deque (дек) на двусвязном списке с методами enqueueFirst(), dequeueFirst(), enqueueLast() и dequeueLast().

Ссылки

[1]. https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Execution_model

[2]. https://developer.mozilla.org/ru/docs/Glossary/Call_stack

[3]. https://srafsan.hashnode.dev/call-stack-in-javascript

[4]. https://dev.to/rajatoberoi/understanding-the-event-loop-callback-queue-and-call-stack-in-javascript-1k7c

[5]. https://www.youtube.com/watch?v=8aGhZQkoFbQ