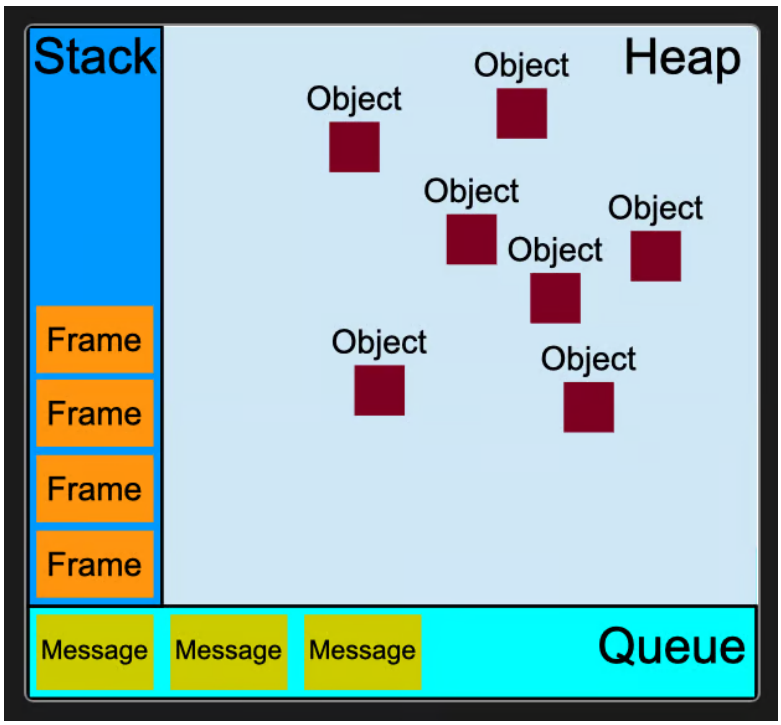
[JavaScript] event loop



- □□□□□□□□□□
- □□□□ Call Stack
  - FILO
- □□□□□□
  - □□□□□□□□□□□□□□□&□□□□□□□□□□□□□□□
- □□□□□ Call Queue
  - FIFO
  - □□ microtasks queue □job queue□
    - Promise
    - process.nextTick(NodeJS)
    - ~~Object.observe(□□)~~
    - MutationObserver
  - □ macrotasks queue□callback queue□
    - script
    - timers□setTimeout/setInterval
    - I/O events
    - user interface events

> TLDR;

javascript-event-loop-call-stack[1]

Call stack

- "□□□"□ each function call is being added to the call stack and removed once it finishes.
- □□/□□□ the stack processes each function call is following the LIFO principle (Last In, First Out).
- Uncaught RangeError: Maximum call stack size exceeded
- The maximum call stack size ranges from 10 to 50 thousand calls

Web APIs

Web APIs

- javascript □□□□□□□□□□□□□□□□□□□□□□□□□□□□ JS □□□□□□□□□□□□ ajax□□□ local storage, service workers□
- alert □□□□ block □□

Callback queue

- setTimeout(fn, 100)□□□ fn □□ Callback queue □
- □□□□□□the callback queue follows the FIFO order (First In, First Out)
- □ call stack □□□□□□□ callback queue □□ task □□ call stack □□□

Event loop

- JavaScript code is being run in a run-to-completion manner, meaning that if the call stack is currently executing some code, the event loop is blocked and won't add any calls from the queue until the stack is empty again
- not to block the call stack by running computation-intensive tasks.

Job queue and asynchronous code

- □□□□□ Callback queue □ The job queue, also known as the promise queue, has priority over the callback queue,
- FIFO order□□□□□□□□ queue□
- await □□□□□□□□□ async □□□□□□□□□□□□□□□□□□□□□□□□□□□□□ promise □□□□□ □□□□□□□□□□□□□□□□□
- async □□□□□□□□□□ promise □□
- □ await □□□□□□□□□□□□□□□□□□□□□□□□□□□□ then □□□

```
async function async1() {
console.log("a"); // 2 - □□□□
await async2(); // await □□ async1 □□□□□□□□□□□□□□□ async2
console.log("b"); // 6
// □□□□
// return Promise.resolve(async2).then(() => console.log("b"));
}
async function async2() {
console.log("c"); // 3
// □□□□
// return Promise.resolve(console.log("c"))□
}
console.log("d"); // 1
setTimeout(function () {
console.log("e"); // 8
}, 0);
async1();
new Promise(function (resolve) {
console.log("f"); // 4
resolve();
}).then(function () {
console.log("g"); // 7
});
console.log("h"); // 5
```

Best Practices

```
function loadImage(src){
  return new Promise((resolve, reject)=>{
  const img = new Image();
  img.onload = () => resolve(img);
  img.onerror = reject;
  img.src = src;
  })
```

```
  })
}

loadImage(src)
  .then(image => resizeImage(image))
  .then(image> applyGrayscaleFilter(image))
  .then(image => addWatermark(image))
```

---

1. https://felixgerschau.com/javascript-event-loop-call-stack

Date: 2024-03-07
Words: 655
Time to read: 3 mins

Newer Older

2024-03-08
[css] tips and tricks

2024-03-06
[JavaScript] closures