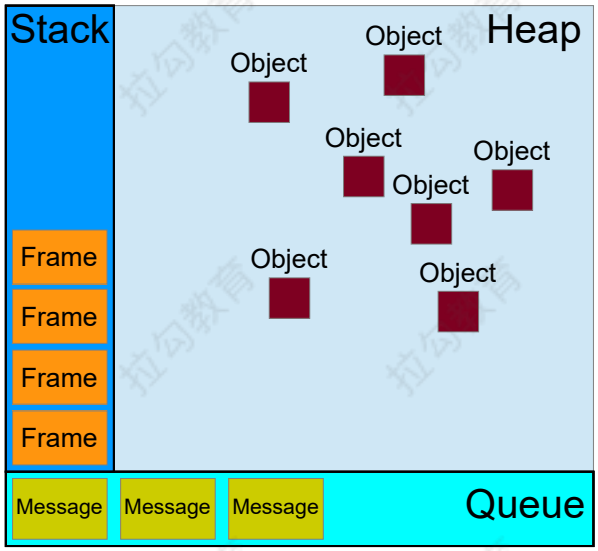
EventLoop、宏任务、微任务

Call Stack



EventLoop

> 事件循环（Event loops）
>
> 每个代理都是由**事件循环**驱动的，事件循环负责收集用事件（包括用户事件以及其他非用户事件等）、对任务进行排队以便在合适的时候执行回调。然后它执行所有处于等待中的 JavaScript 任务，然后是微任务，然后在开始下一次循环之前执行一些必要的渲染和绘制操作。

```
console.log('start')

setTimeout(function () {
  console.log(1)
}, 0)
```

```
// 事件循环，主线程
while (macroQueue.waitForMessage()) {
  // 1. 执行完调用栈上当前的宏任务(同步任务)
  // call stack


  // 2. 遍历微任务队列，把微任务队里上的所有任务都执行完毕(清空微任务队列)
  // 微任务又可以往微任务队列中添加微任务
  for (let i = 0; i < microQueue.length; i++) {
    // 获取并执行下一个微任务(先进先出)
    microQueue[i].processNextMessage()
```

```
  }

  // 3. 渲染（渲染线程）

  // 4. 从宏任务队列中取 一个 任务，进入下一个消息循环
  macroQueue.processNextMessage();
}
```

任务 vs 微任务

一个**任务**就是指计划由标准机制来执行的任何 JavaScript，如程序的初始化、事件触发的回调等。 除了使用事件，你还可以使用 `setTimeout()` 或者 `setInterval()` 来添加任务。

任务队列和微任务队列的区别很简单，但却很重要：

- 当执行来自任务队列中的任务时，在每一次新的事件循环开始迭代的时候运行时都会执行队列中的每个任务。在每次迭代开始之后加入到队列中的任务需要**在下一次迭代开始之后才会被执行**.
- 每次当一个任务退出且执行上下文为空的时候，微任务队列中的每一个微任务会依次被执行。不同的是它会等到微任务队列为空才会停止执行——即使中途有微任务加入。换句话说，微任务可以添加新的微任务到队列中，并在下一个任务开始执行之前且当前事件循环结束之前执行完所有的微任务。

https://developer.mozilla.org/zh-CN/docs/Web/API/HTML_DOM_API/Microtask_guide/In_depth

产生宏任务的方式

- script 中的代码块
- setTimeout()
- setInterval()
- setImmediate() (非标准，IE 和 Node.js 中支持)
- 注册事件

产生微任务的方式

- Promise
- MutationObserver
- queueMicrotask()

```
queueMicrotask(() => {
  console.log('微任务')
})
```

何时使用微任务

微任务的执行时机，晚于当前本轮事件循环的 Call Stack 中的代码(宏任务)，早于事件处理函数和定时器的回调函数。

使用微任务的最主要原因简单归纳为：

- 减少操作中用户可感知到的延迟
- 确保任务顺序的一致性，即便当结果或数据是同步可用的

- 批量操作的优化

确保任务顺序的一致性

```
customElement.prototype.getData = url => {
  if (this.cache[url]) {
    this.data = this.cache[url];
    this.dispatchEvent(new Event("load"));
  } else {
    fetch(url).then(result => result.arrayBuffer()).then(data => {
      this.cache[url] = data;
      this.data = data;
      this.dispatchEvent(new Event("load"));
    )};
  }
};

element.addEventListener("load", () => console.log("Loaded data"));
console.log("Fetching data...");
element.getData();
console.log("Data fetched");
```

数据未缓存的结果（左） vs. 缓存中有数据的结果

数据未缓存	数据已缓存
Fetching data Data fetched Loaded data	Fetching data Loaded data Data fetched

```
customElement.prototype.getData = url => {
  if (this.cache[url]) {
    queueMicrotask(() => {
      this.data = this.cache[url];
      this.dispatchEvent(new Event("load"));
    });
  } else {
    fetch(url).then(result => result.arrayBuffer()).then(data => {
      this.cache[url] = data;
      this.data = data;
      this.dispatchEvent(new Event("load"));
    )};
  }
};
```

批量操作

```
let messageQueue = []
let sendMessage = message => {
  messageQueue.push(message)
  if (messageQueue.length === 1) {
    queueMicrotask(() => {
      const json = JSON.stringify(messageQueue);
      messageQueue.length = 0;
```

```
      // fetch("url-of-receiver", json);
      console.log(json)
    });
  }
};

sendMessage('刘备')
sendMessage('关羽')
sendMessage('曹操')
```

参考资料

- 并发模型与事件循环
- Microtasks and the JavaScript runtime environment
- 在 JavaScript 中通过 queueMicrotask() 使用微任务
- https://developer.mozilla.org/zh-CN/docs/Web/API/HTML_DOM_API/Microtask_guide/In_depth
- Event loops