高频面试题-事件循环

微信公众号：小狮子前端
Vue 回复【事件循环】

宏任务 和 微任务 你又知多少

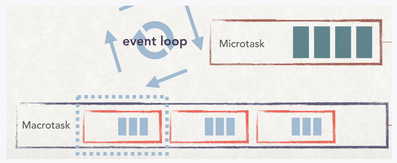
先来介绍宏任务

- 页面中大部分任务都在 主线程 上执行
- 渲染事件（如解析 DOM、计算布局、绘制）
- 用户交互事件（如鼠标点击、滚动页面、放大缩小等）
- JavaScript 脚本执行事件
- 网络请求完成、文件读写完成事件

宏任务有什么不足点

添加事件是由系统操作的，JavaScript 代码不能准确掌控任务要添加到队列中的位置。

那么，控制不了任务在消息队列中的位置，所以很难控制开始执行任务的时间。



再来介绍微任务

微任务就是一个需要异步执行的函数，执行时机是在主函数执行结束之后、当前宏任务结束之前。

从V8引擎层面来看微任务

当 JavaScript 执行一段脚本的时候，V8 会为其创建一个全局执行上下文，在创建全局执行上下文的同时，V8 引擎也会在内部创建一个 微任务队列。

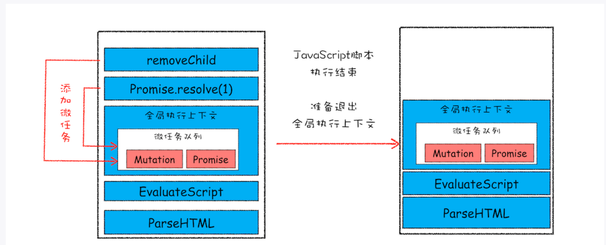
微任务的产生时机

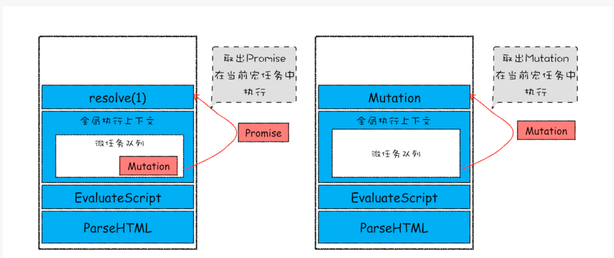
第一种方式是使用 MutationObserver 监控某个 DOM 节点，然后再通过 JavaScript 来修改这个节点，或者为这个节点添加、删除部分子节点，当 DOM 节点发生变化时，就会产生 DOM 变化记录的微任务。

第二种方式是使用 Promise，当调用 Promise.resolve() 或者 Promise.reject() 的时候，也会产生微任务。

微任务何时被执行

通常情况下，在当前宏任务中的 JavaScript 快执行完成时，也就在 JavaScript 引擎准备退出全局执行上下文并清空调用栈的时候，JavaScript 引擎会检查全局执行上下文中的微任务队列，然后按照顺序执行队列中的微任务。WHATWG 把执行微任务的时间点称为 检查点。





宏任务和微任务关系（总结）

微任务和宏任务是绑定的，每个宏任务在执行时，会创建自己的微任务队列。

宏任务和微任务之间的关系。 因为在微任务中产生的宏任务也是要插入到消息队列或者是延迟队列的末尾的，这肯定是需要下一次事件循环才有可能被执行的，而微任务在这一次的事件循环之前就会被执行。 因此，无论什么情况下，微任务都早于宏任务执行。

微任务的执行时长会影响到当前宏任务的时长。比如一个宏任务在执行过程中，产生了 100 个微任务，执行每个微任务的时间是 10 毫秒，那么执行这 100 个微任务的时间就是 1000 毫秒，也可以说这 100 个微任务让宏任务的执行时间延长了 1000 毫秒。所以你在写代码的时候一定要注意控制微任务的执行时长。

带你了解WebAPI：setTimeout

它就是一个定时器，用来指定某个函数在多少毫秒之后执行。它会返回一个整数，表示定时器的编号，同时你还可以通过该编号来取消这个定时器。

通过定时器设置回调函数有点特别，它们需要在指定的时间间隔内被调用，但消息队列中的任务是按照顺序执行的，因此，不能将定时器的回调函数直接添加到消息队列中。于是 Chrome 又维护了一个另外一个消息队列，用来存放延迟执行的任务列表。（这里我就叫做 延迟队列 了）

使用 setTimeout 的一些注意事项

1. 如果当前任务执行时间过久，会影响定时器任务的执行
2. 如果 setTimeout 存在嵌套调用，那么系统会设置最短时间间隔为 4 毫秒
3. 未激活的页面，setTimeout 执行最小间隔是 1000 毫秒
4. 延时执行时间有最大值
5. 使用 setTimeout 设置的回调函数中的 this 不符合直觉

如何安全退出

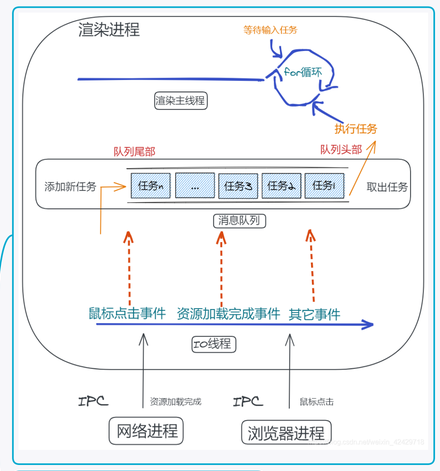
确定要退出当前页面时，页面主线程会设置一个退出标志的变量，在每次执行完一个任务时，判断是否有设置退出标志。如果设置了，那么就直接中断当前的所有任务，退出线程。

消息队列中的任务类型

消息队列中的任务类型包含了很多内部消息类型，如输入事件（鼠标滚动、点击、移动）、微任务、文件读写、WebSocket、JavaScript 定时器等等。

除此之外，消息队列中还包含了很多与页面相关的事件，如 JavaScript 执行、解析 DOM、样式计算、布局计算、CSS 动画等。

处理其它线程发送过来的任务

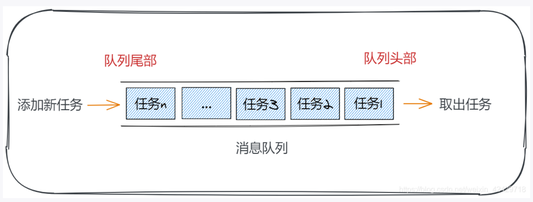


从图中可以看出，渲染进程专门有一个 IO 线程用来接收其他进程传进来的消息，接收到消息之后，会将这些消息组装成任务发送给渲染主线程，后续的步骤就与上文一样了，这里就不再赘述了。

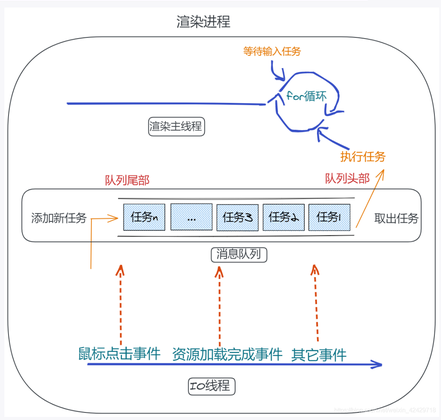
处理其它线程发送过来的任务

引入消息队列

消息队列是一种数据结构，可以存放要执行的任务。它符合队列 “先进先出” 的特点，也就是说要添加任务的话，添加到队列的尾部；要取出任务的话，从队列头部去取。



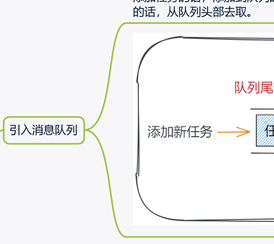
有了消息队列机制后，我们就可以改造一下之前的模型



单线程处理安排好的任务

将任务按照顺序依次执行，等所有任务执行完

引入循环机制



要想在线程运行过程中，能接收并执行新的任务，就需要采用事件循环机制。

WHATWG 规范定义事件循环机制

- 先从多个消息队列中选出一个最老的任务，这个任务称为 oldestTask
- 然后循环系统记录任务开始执行的时间，并把这个 oldestTask 设置为当前正在执行的任务
- 当任务执行完成之后，删除当前正在执行的任务，并从对应的消息队列中删除掉这个 oldestTask
- 最后统计执行完成的时长等信息

分享工具

- 作图工具：excalidraw
- Loupe 可视化工具

可以帮助您了解JavaScript的调用堆栈/事件循环/回调队列如何相互影响)工具来了解代码的执行情况