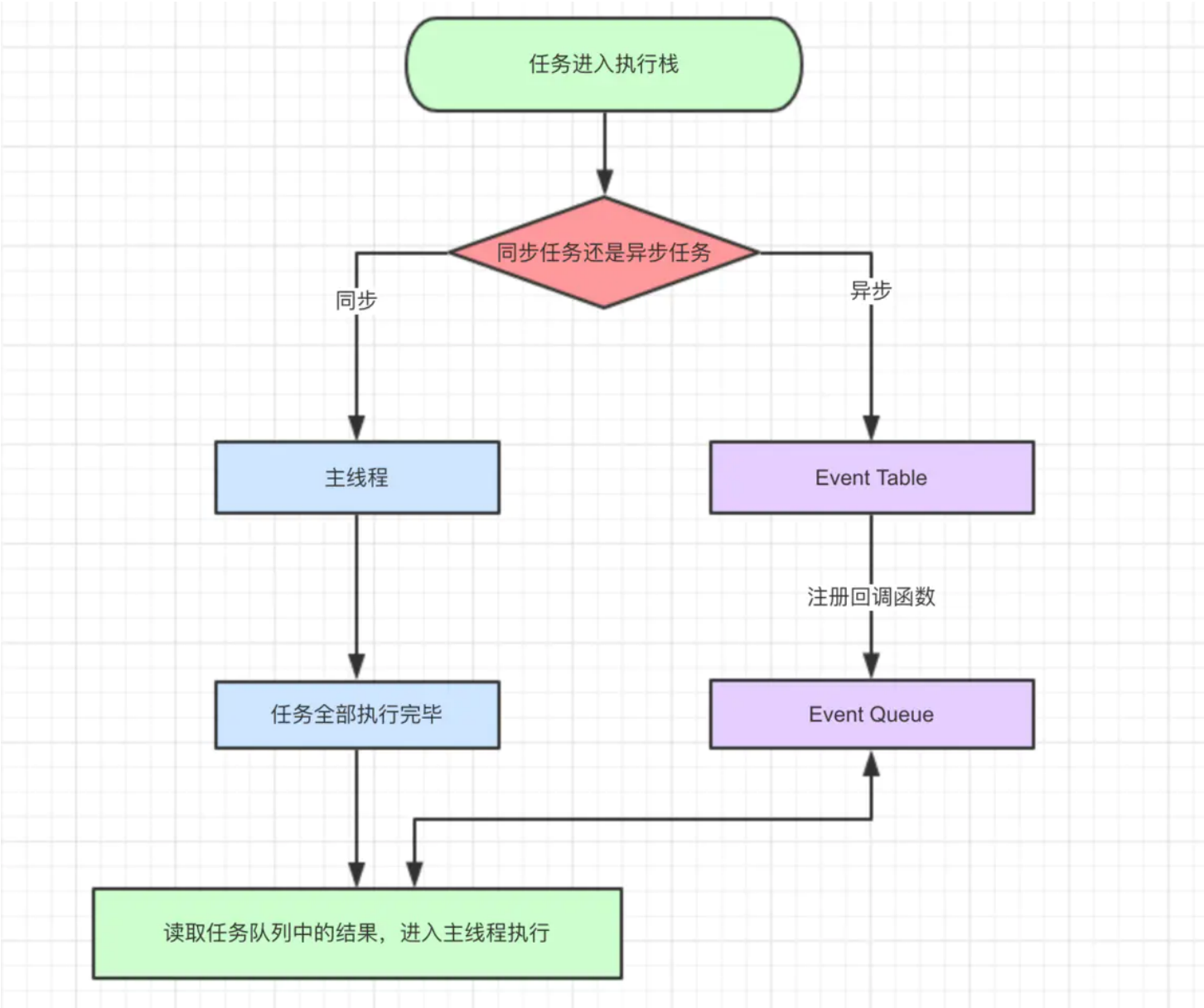
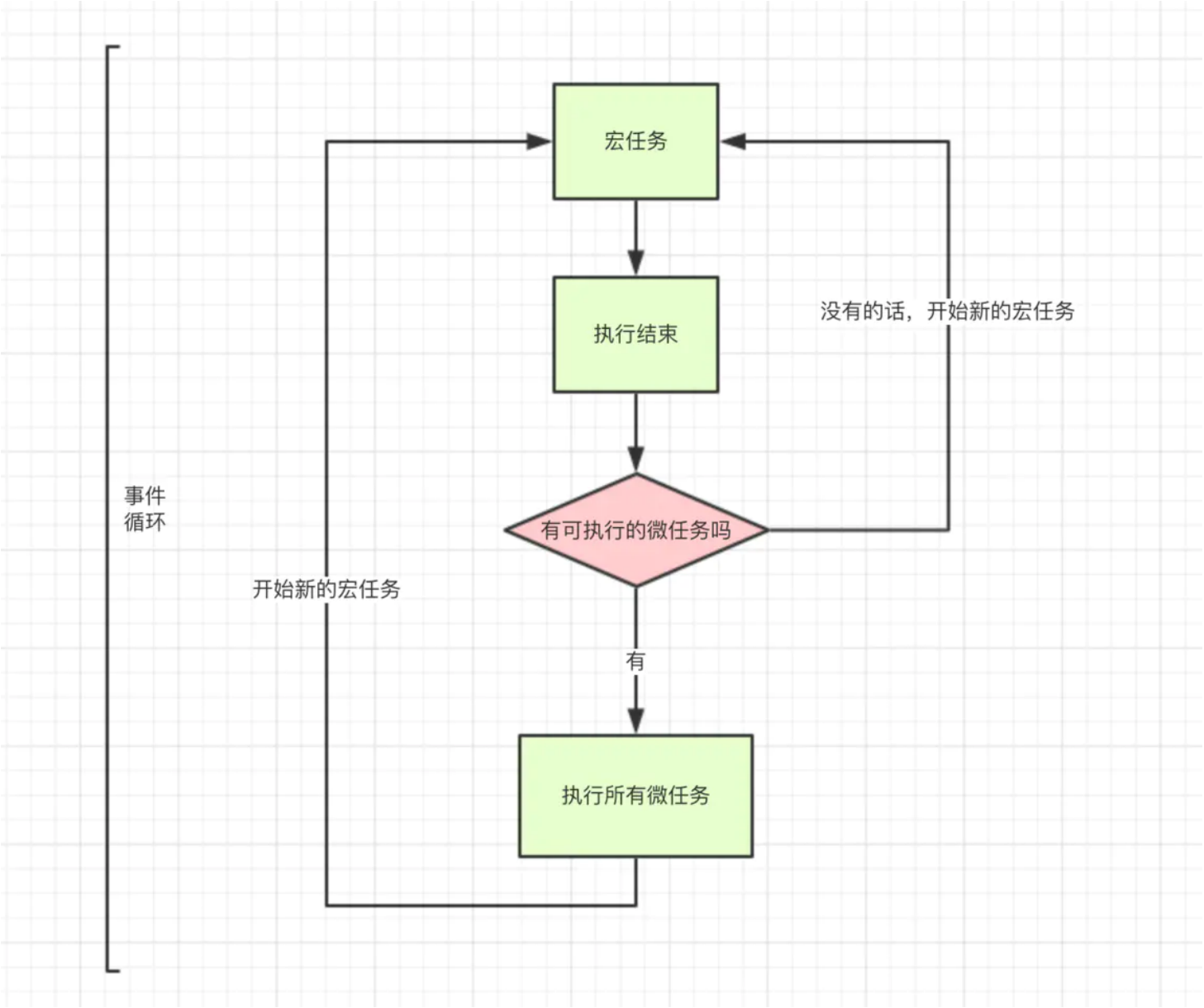
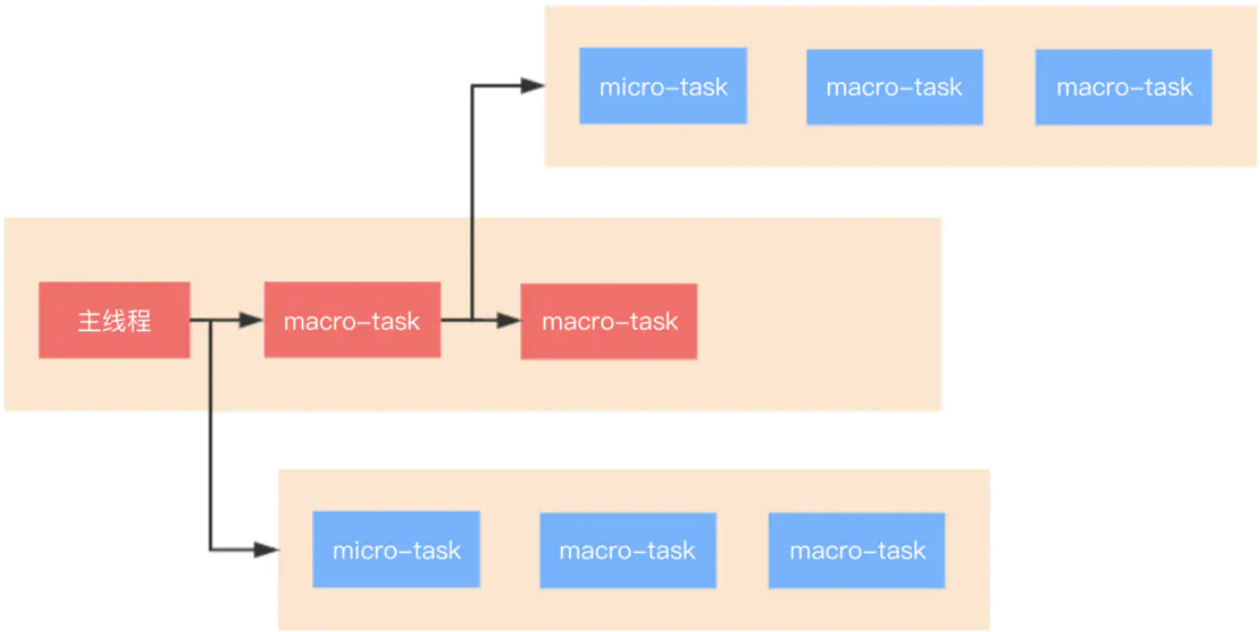
Event-Loop

浏览器环境:



Event-loop:


宏任务
执行结束
没有的话，开始新的宏任务
事件
循环
有可执行的微任务吗
开始新的宏任务
有
执行所有微任务

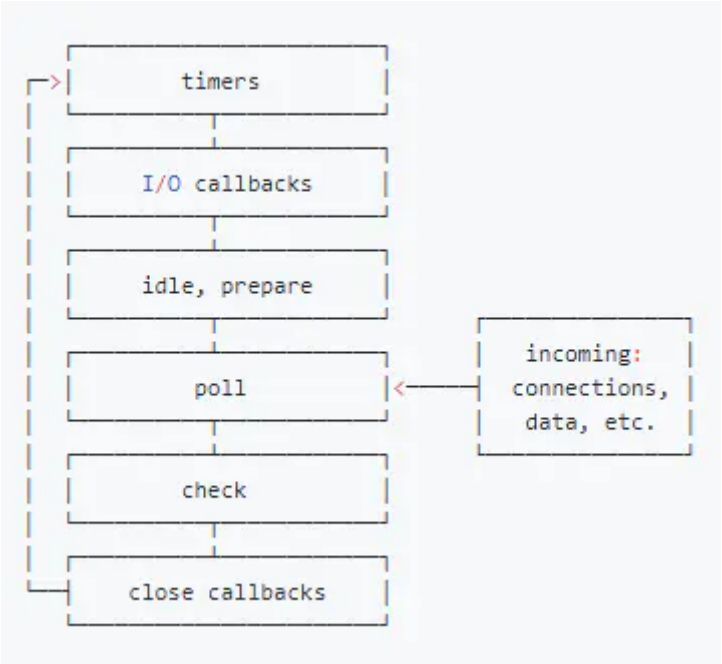

micro-task
macro-task
macro-task
主线程
macro-task
macro-task
micro-task
macro-task
macro-task

执行宏任务，然后执行该宏任务产生的微任务，若微任务在执行过程中产生了新的微任务，则继续执行微任务，微任务执行完毕后，再回到宏任务中进行下一轮循环

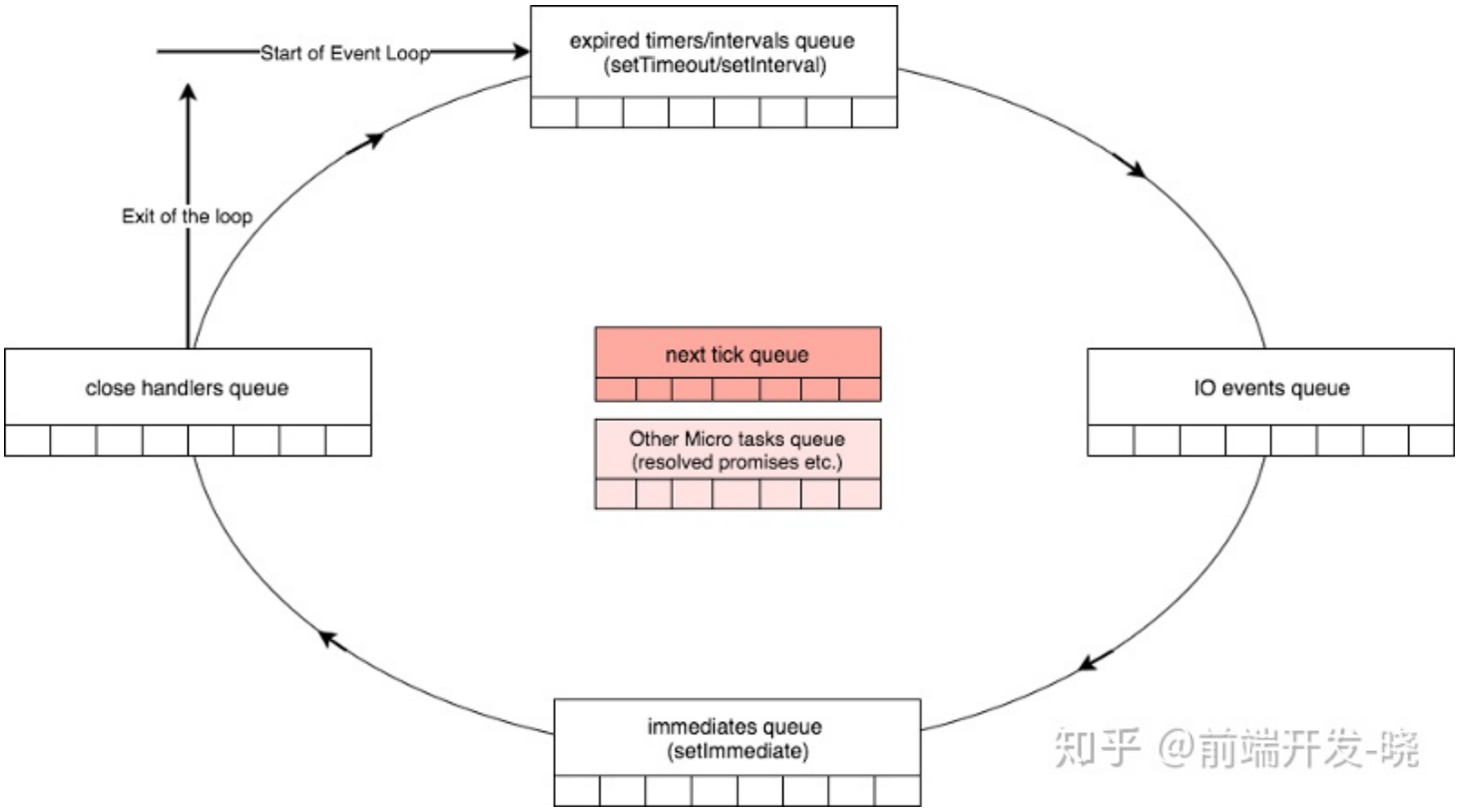
浏览器环境下的宏任务和微任务：

- macro-task: 主线程、setTimeout、setInterval、IO、UI渲染
- micro-task: promise

Node环境：



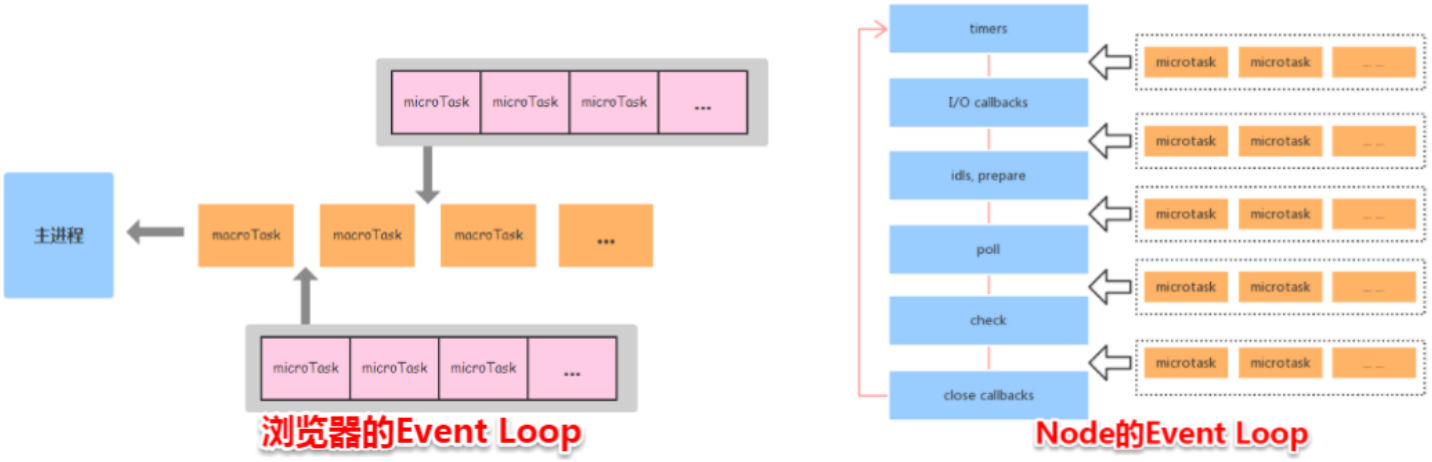
Node.js事件循环讲解



在 NodeJS 中不止一个队列，不同类型的事件在它们自己的队列中入队。
总共有四种类型的队列(宏任务):

- Expired timers and intervals queue: setTimeout和setInterval的回调
- IO Events Queue：完成的 I/O 事件的回调
- Immediate queue：setImmediate的回调
- Close Handlers Queue：任何一个 close 事件的回调(例如socket.on('close', callback))。
  在处理完一个类型的队列后，移向处理下一个队列之前，
  事件循环将会检查这两个中间阶段是否有微任务（process.nextTick和promise回调，且nextTick queue比promise queue有着更高的优先级）要处理，如果有，事件循环会立马开始处理它们。一旦微任务队列为空，事件循环就移到下一个阶段

浏览器环境和Node环境事件循环的异同：



浏览器环境下，microtask的任务队列是每个macrotask执行完之后执行。而在Node.js中，microtask会在事件循环的各个阶段之间执行，也就是一个阶段执行完毕，就会去执行microtask队列的任务。

经典面试题：