28：灵魂拷问：Node中的Event-Loop与浏览器有何不同？

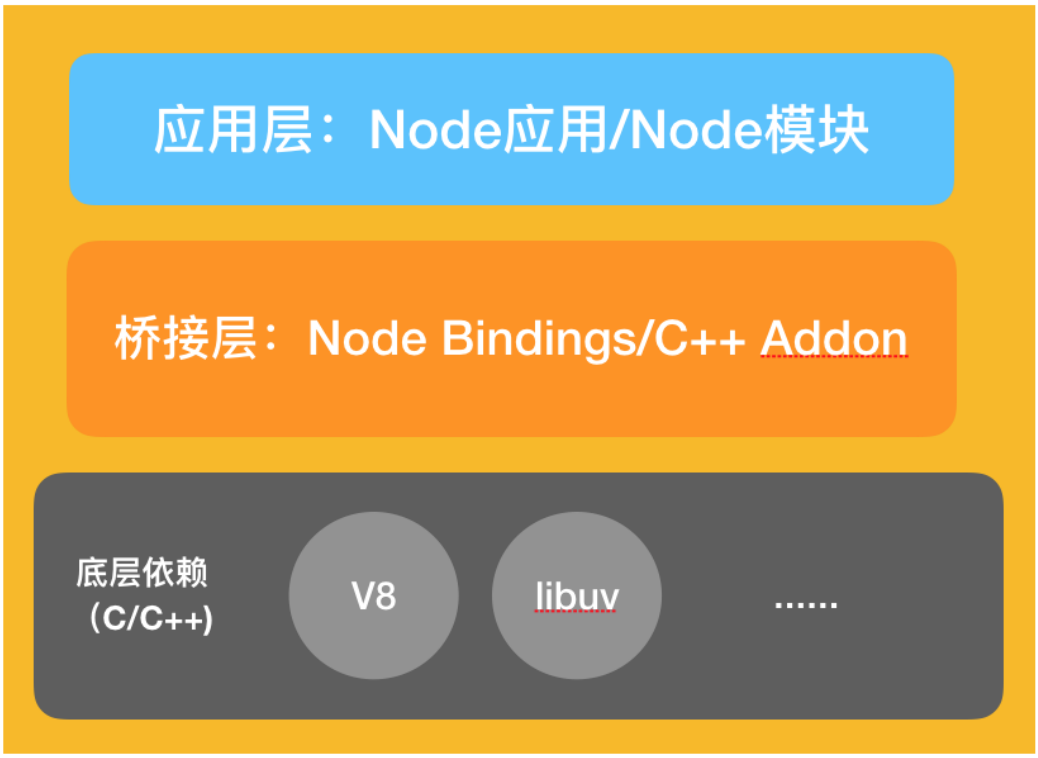
更新时间：2020-05-26 14:25:51

> 勤能补拙是良训，一分辛劳一分才。——华罗庚

如标题所示，这是一道大厂面试官普遍钟爱的面试题。要想答出 Node 中的 Event-Loop 和浏览器有啥区别，首先你得能说清楚，Node 中的 Event-Loop 本身是怎么一回事。

Node技术架构分析-认识 libuv

这里我为大家画了一张简化的 Node 架构图：



Node整体上由这三部分组成：

应用层：这一层就是大家最熟悉的 Node.js 代码，包括 Node 应用以及一些标准库。

桥接层：Node 底层是用 C++ 来实现的。桥接层负责封装底层依赖的 C++ 模块的能力，将其简化为 API 向应用层提供服务。

底层依赖：这里就是最最底层的 C++ 库了，支撑 Node 运行的最基本能力在此汇聚。其中需要特别引起大家注意的就是 V8 和 libuv：

- V8 是 JS 的运行引擎，它负责把 JavaScript 代码转换成 C++，然后去跑这层 C++ 代码。
- **libuv：它对跨平台的异步I/O能力进行封装，同时也是我们本节的主角：Node 中的事件循环就是由 libuv 来初始化的。**

注意哈：这里第一个区别来了——浏览器的 Event-Loop 由各个浏览器自己实现；而 Node 的 Event-Loop 由 libuv 来实现。

libuv中的 Event-Loop 实现

libuv 主导循环机制共有六个循环阶段。这里我引用 Node 官方（出处：https://nodejs.org/zh-cn/docs/guides/event-loop-timers-and-nexttick/）的一张图给大家作说明：

（注：Node 官方给出的这张图非常值得参考，不过不建议大家直接通过阅读其官方文档来理解事件循环，一些表达还是会相对比较生涩，打击积极性）。

我们先来瞅瞅这六个阶段各是处理什么任务的：

- **timers阶段**：执行 setTimeout 和 setInterval 中定义的回调；
- **pending callbacks**：直译过来是“被挂起的回调”，如果网络I/O或者文件I/O的过程中出现了错误，就会在这个阶段处理错误的回调（比较少见，可以略过）；
- **idle, prepare**：仅系统内部使用。这个阶段我们开发者不需要操心。（可以略过）；
- **poll （轮询阶段）**：重点阶段，这个阶段会执行I/O回调，同时还会检查定时器是否到期；
- **check（检查阶段）**：处理 setImmediate 中定义的回调；
- **close callbacks**：处理一些“关闭”的回调，比如 `socket.on('close', ...)` 就会在这个阶段被触发。

宏任务与微任务

和浏览器中一样，Node 世界里也有宏任务与微任务之分。划分依据与我们上文描述的其实是一致的：

> 常见的 macro-task 比如： setTimeout、setInterval、 setImmediate、 script（整体代码）、I/O 操作、UI 渲染等。
>
> 常见的 micro-task 比如: process.nextTick、Promise、MutationObserver 等

需要注意的是，setImmediate 和 process.nextTick 是 Node 独有的，在本节各位会有充分的机会和它们打交道。

一起走一遍 Node 中的事件循环流程

在这六个阶段中，大家需要重点关注的就是 **timers、poll 和 check** 这 三个阶段，相关的命题也基本上是围绕它们来做文章。不过在进行考点点拨之前，我们还是要把整个循环的流程给走一遍：

1. 执行全局的 Script 代码（与浏览器无差）；
2. 把微任务队列清空：注意，Node 清空微任务队列的手法比较特别。在浏览器中，我们只有一个微任务队列需要接受处理；但在 Node 中，有两类微任务队列：next-tick 队列和其它队列。其中这个 next-tick 队列，专门用来收敛 process.nextTick 派发的异步任务。**在清空队列时，优先清空 next-tick 队列中的任务，随后才会清空其它微任务**；
3. 开始执行 macro-task（宏任务）。注意，Node 执行宏任务的方式与浏览器不同：在浏览器中，我们每次出队并执行一个宏任务；而在 Node 中，我们每次会尝试清空当前阶段对应宏任务队列里的所有任务（除非达到了系统限制）；
4. 步骤3开始，会进入 3 -> 2 -> 3 -> 2...的循环（整体过程如下所示）：

```
micro-task-queue ----> timers-queue
                   |
                   |
micro-task-queue ----> pending-queue
                   |
                   |
micro-task-queue ---->  polling-queue
                   |
                   |
micro-task-queue ---->  check-queue
                   |
                   |
micro-task-queue ---->  close-queue
                   |
                   |
micro-task-queue ----> timers-queue

......
```

整体来看，Node 中每次执行异步任务都是以批量的形式，“一队一队”地执行。循环形式为：宏任务队列 -> 微任务队列 -> 宏任务队列 —> 微任务队列... 这样交替进行。

经过咱们上面这一通讲，相信把 Node 和浏览器之间不同的 Event-Loop 机制捋清楚，对现在的你来说已经不是什么难事了。

不过，可别高兴得太早。事件循环这块，比起问答题，更常见的是编码阅读题。咱们下面就一起来通过一系列的真题巩固一下认知。

}

27 浏览器中的 Event-Loop

29 Node事件循环命题思路剖析