事件循环3

欢迎回到事件循环系列文章~，在第一章中，我们对 NodeJs 的事件循环以及事件循环的各个阶段有了一个总体的了解。在第二章中，我们学习了 timer、immediates 队列在事件循环中的特性和运作机制。在本章中，我们将要研讨事件循环机制是如何处理 resolved/rejected promise 事件，以及 next tick 事件（回调函数）。若是你对 promise 还不熟悉，建议你先学习一下 promise 的语法，它非常接地气。

由于下文中的事件和回调函数是等价的，所以下文中提到的事件就是回调函数。

**原生 promise**

原生 promise 被看做是一个微任务 ( microtask ) ，并且存储于微任务事件队列 (microtask queue) 中。该事件队列在 next tick 队列执行完毕后执行。

请看下面的例子：

上述的例子，进行了以下操作：

1. 往微任务队列 ( microtask queue ) 中加入了 5 个微任务事件 ( microtask )，需要注意的是，此处我们加入的微任务都是 resolved 回调 ( 已完成 )
2. 使用 setImmediate 往 immediate queue 队列中加入了 2 个事件
3. 往 nextTick 队列中加入了 3个 事件
4. 创建了一个 0 秒过期的定时器，为 timer queue 队列中加入了一个 timer 事件
5. 往 immediate queue 队列中又加入了 2 个 immediate 事件

下面是事件循环的执行过程：

1. 事件循环机制首先会去检查 next tick 队列，然后 Node 将会处理其中的事件，直到队列为空。( 上述例子中有 3 个 next tick 事件 )
2. 事件机制将会检测 microtask queue 队列，执行其中的事件，直到队列为空。( 上述例子中有 5 个 promise resolved 事件 )
3. 当处理 microtask queue 队列中的 promise resolved 事件 ( 回调 ) 时， 一个 next tick 事件被加入了 next tick queue 队列 （ promise resolved 回调事件中调用了 process.nextTick 添加 ）。
4. 当 microtask queue 队列中的所有事件都执行完毕后，由于之前往 next tick 队列中又塞入了一个事件，这会事件机制会再去执行完 next tick 队列中的所有事件。
5. 当 microtask queue 和 next tick queue 队列中都没有额外事件要处理后，事件循环将会转入 timer queue 阶段（上述例子中，此时有 5 个到期的 timer 事件将被依次执行）
6. 执行完 timer queue 队列后，事件循环将会转入 I/O queue 队列，由于本例中，并没有 I/O queue 事件，故而事件循环将会转入 immediate 队列（ 本例中，immediate queue 队列中一共有 4 个事件，将会被依次执行 ）
7. 最后，整个事件循环都没有可供执行的事件，程序退出

那么上文中，为何我不厌其烦地提及 promises microtask 而不是 microtasks ，请看下文。

最主要是因为 resolved/rejected promise 事件和 process.nextTick 事件同属于 microtasks。

我们来看一下上面例子的执行结果

**Q and Bluebird**

现在我们知道了，原生的 resolve/reject 回调函数 ( 即 promise 事件 ) ，在 Node 中是被当做 microtask 处理的，且当事件循环机制从一个事件阶段转移到下一个阶段的当中才会被执行。那么 Q 和 Bluebird 这类第三方库是如何实现 Promise 的呢 ?