事件循环2

在上一篇文章中，我们对 NodeJs 的事件循环机制已经有了一些大体的了解。在本章中，我会结合例子，对前文中提到过的 timers, immediates 和 nextTick 队列( process.nextTick 产生 ) 做出详细的分析

**Next Tick 队列**

首选再看一下事件循环的图示

Next tick 队列与其他 4 个主队列单独区分展示的原因是因为，Next Tick 队列是由 Node 本身提供的而非 libuv 的原生功能。

当事件循环机制转入每一个主要事件循环阶段前时 ( timers queue, IO events queue, immediates queue, close queue )，都会先去检查 nextTick 队列中是否有待处理的事件，直到处理完 nextTick 队列中的事件之后才会转入将要转入的主要事件循环阶段。

这就引入了一个问题，当我们频繁或是递归地使用 process.nextTick 方法往 nextTick 队列中塞入事件时，将会引起 I/O 或是其他队列中的事件处于饿死状态，永远无法得到执行。我们举一个简单的例子，验证这个问题。

我们可以看到，程序的输出结果，始终是无线递归的 nextTick 中的事件输出，而 setTimeout, setImmediate 和 fs.readFile ( file I/O )则永远得不到执行。我们可以为 addNextTickRecurs 函数设定一个递归值，则其余事件循环阶段的事件将在 nextTick 执行完后得到执行。

**Timers queue ( 定时器队列 )**

当我们使用 setTimeout 或 setInterval 创建一个定时器时，NodeJs 会将这个定时器加入由 libuv 提供的计数器堆。当事件循环执行到 timer queue 阶段，NodeJs 会去计时器堆里找到期的 timer/intervals 定时器，并立即执行其绑定的回调函数。若是有超过一个到期的定时器，则按照他们当初设定的顺序依次执行。

一个设有特定时限的 timer/interval 定时器，在到期后并不能保证一定会被立即执行。这和系统的性能也有一定的关系，因为在执行定时器的回调函数之前，Node 会去检查定时器是否到期，这会消耗一定的时间。同样地，当前事件循环中正在执行的进程，也会对定时器的执行产生影响。过期时间只能保证，定时器不会在给定的到期事件内触发。下面我们举一个简单的例子验证一下。

上面的程序会新建一个过期时间为 1000 毫秒的定时器，并监测回调函数执行时候的时间。