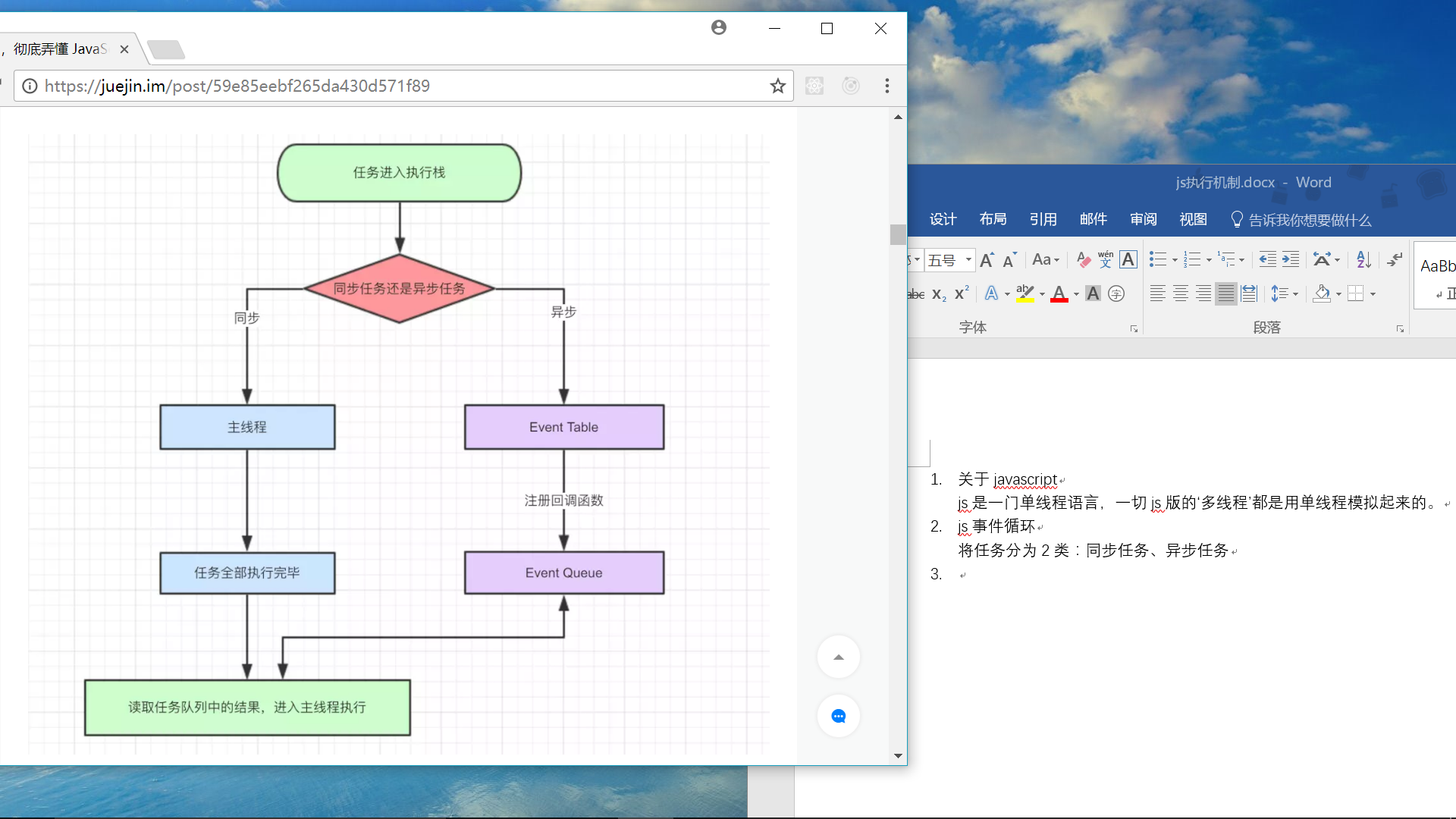
1. 关于javascript

js是一门单线程语言，一切js版的‘多线程’都是用单线程模拟起来的。

1. js事件循环

将任务分为2类：同步任务、异步任务



同步任务进入主线程，异步任务进入Event Table并注册函数

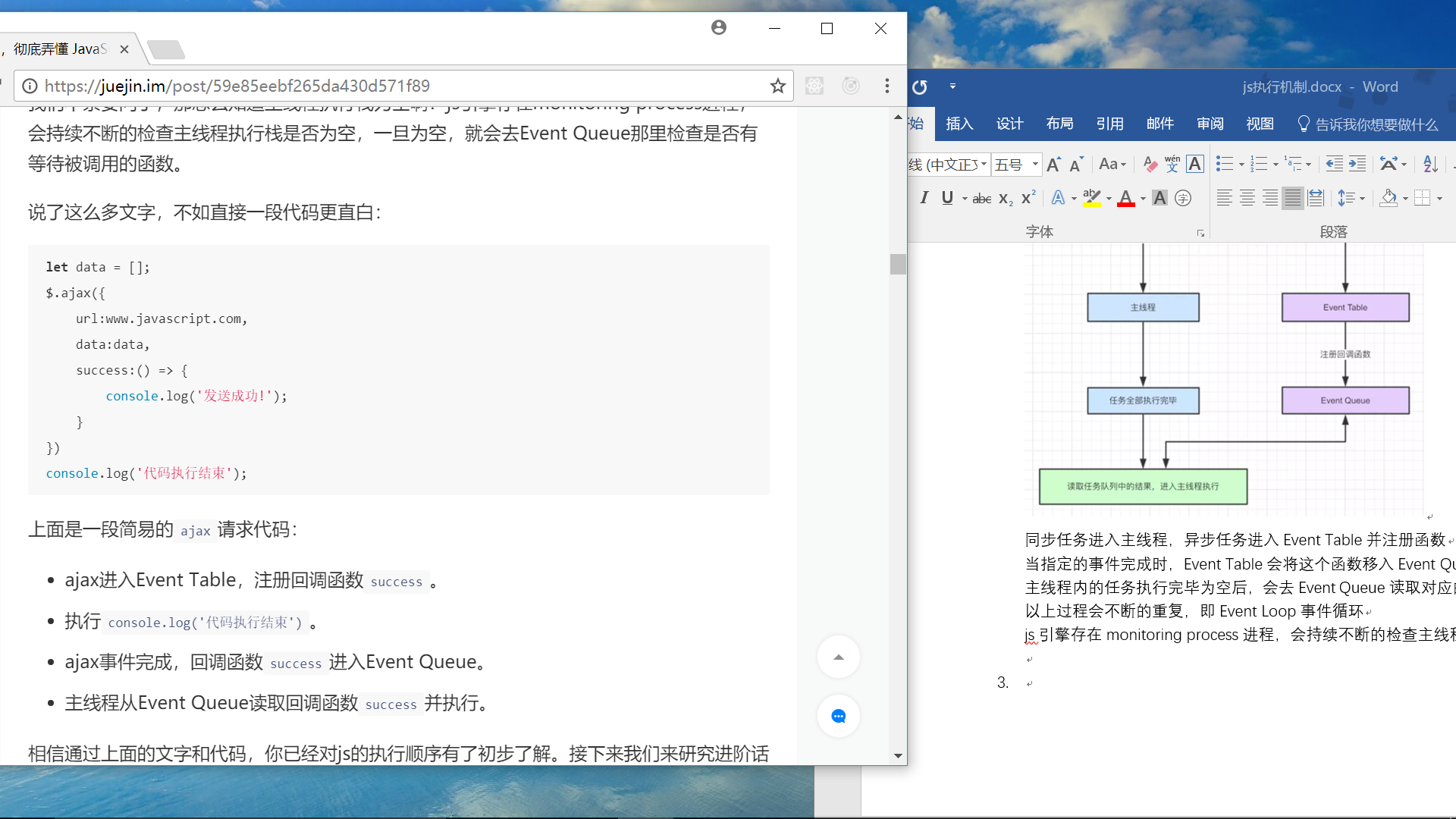
当指定的事件完成时，Event Table会将这个函数移入Event Queue

主线程内的任务执行完毕为空后，会去Event Queue读取对应的函数，进入主线程运行

以上过程会不断的重复，即Event Loop事件循环

js引擎存在monitoring process进程，会持续不断的检查主线程执行栈是否为空

例



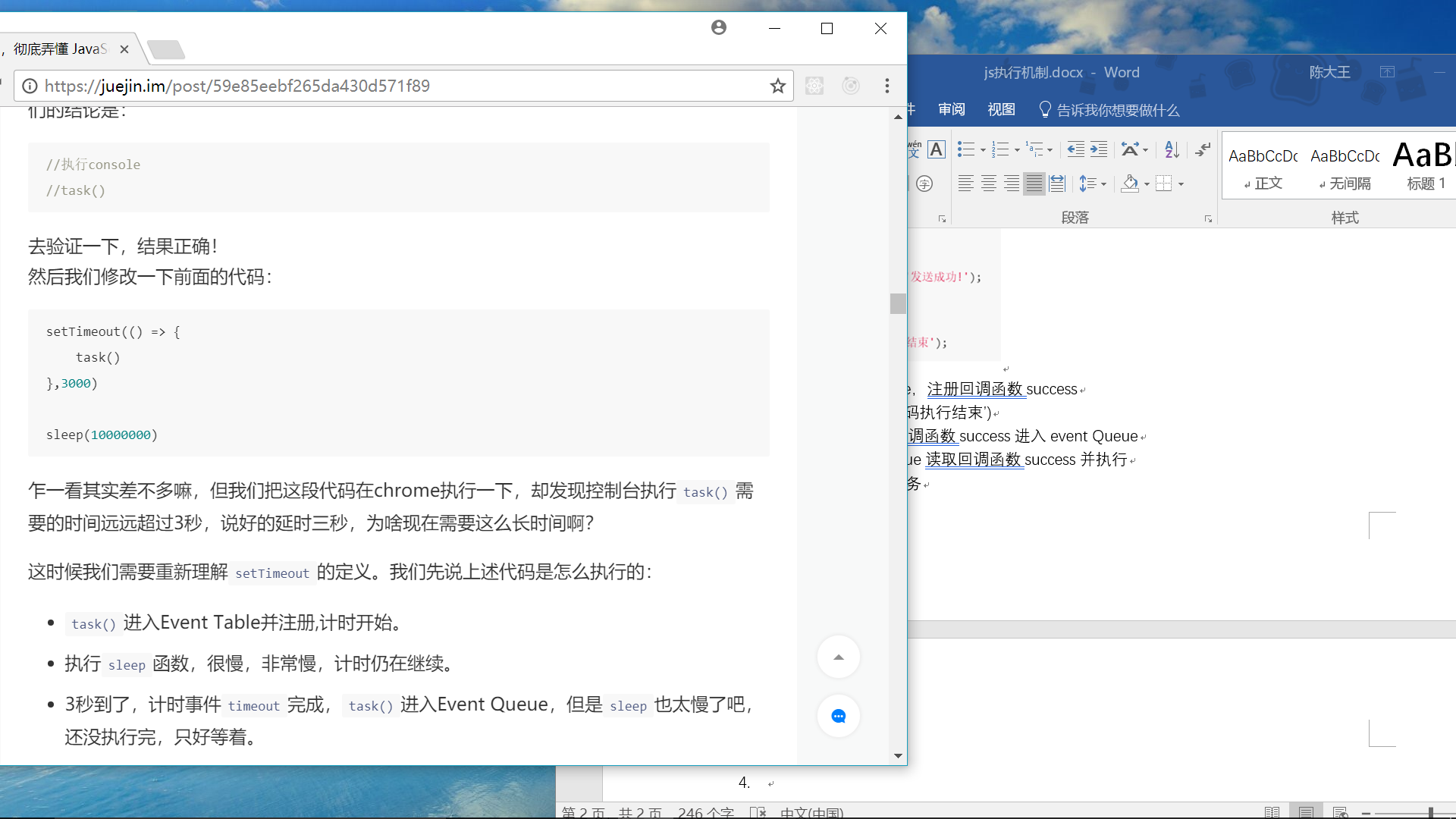
ajax进入Event Table，注册回调函数success

执行console.log(‘代码执行结束’)

ajax事件完成后，回调函数success进入event Queue

主线程从Event Queue读取回调函数success并执行

1. setTimeout 异步任务



task()进入Event Table并注册，计时开始

执行sleep()

3秒到了，计时事件完成，task()进入Event Queue，此时sleep()仍在执行

sleep()执行完毕

task()从Event Queue进入主线程并执行

setTimeout(fn,0)的含义是，指定某个任务在主线程最早可得的空闲执行，不用再等多少秒，只要主线程执行栈内的同步任务全部执行完就可以马上执行

1. setInterval

setInterval会每隔指定时间将注册的函数置入Event Queue

setInterval(fn,ms)不是每过ms秒就执行一次fn，而是每过ms秒，就会有fn进入Event Queue

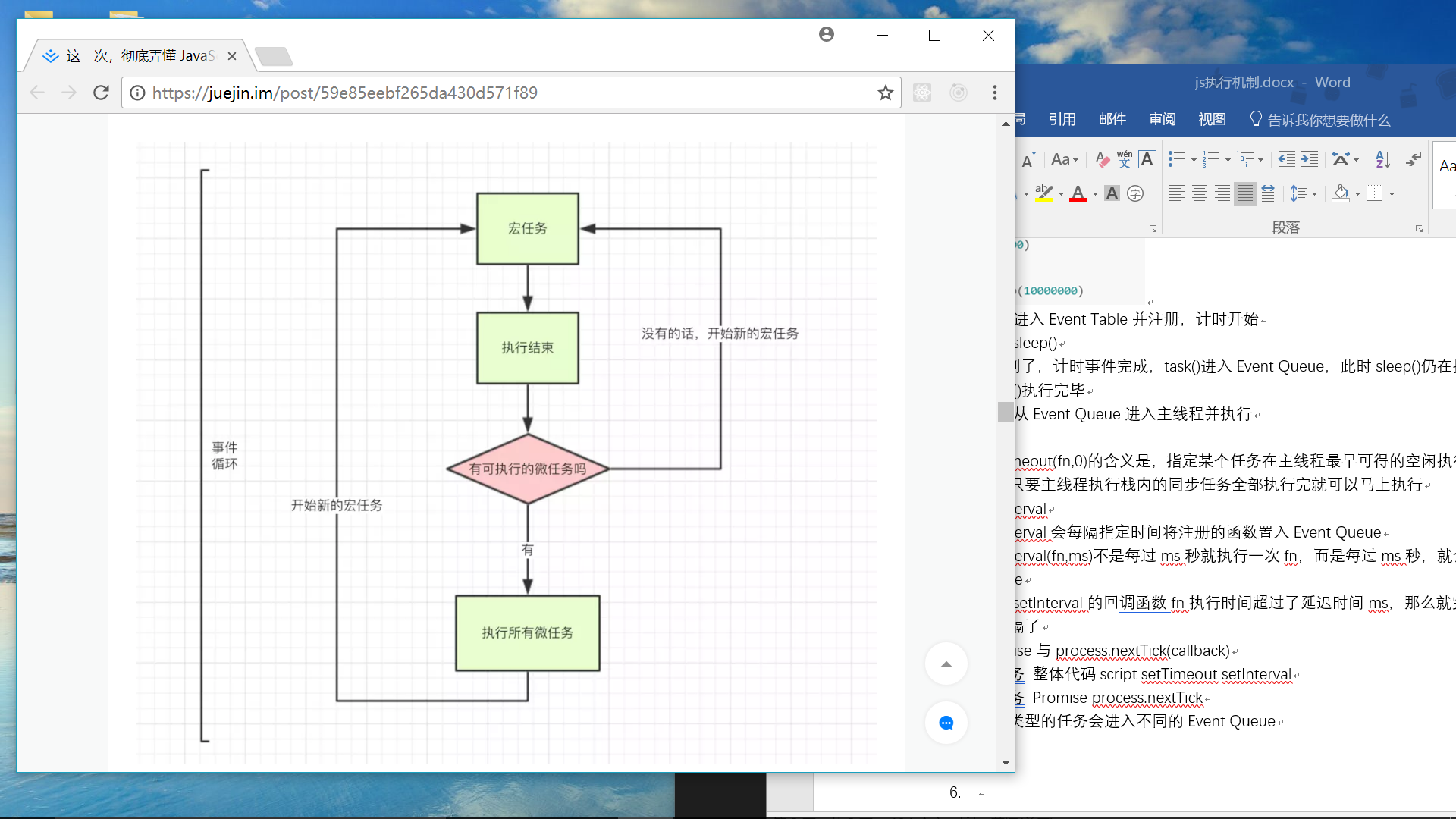
一旦setInterval的回调函数fn执行时间超过了延迟时间ms，那么就完全看不出来有时间间隔了

1. Promise与process.nextTick(callback)

宏任务 整体代码script setTimeout setInterval

微任务 Promise process.nextTick

不同类型的任务会进入不同的Event Queue



例



先遇到setTimeout，将其回调函数注册后分发到宏任务Event Queue

接下来遇到Promise，new Promise立即执行，then函数分发到微任务Event Queue

遇到console.log(‘console’)立即执行

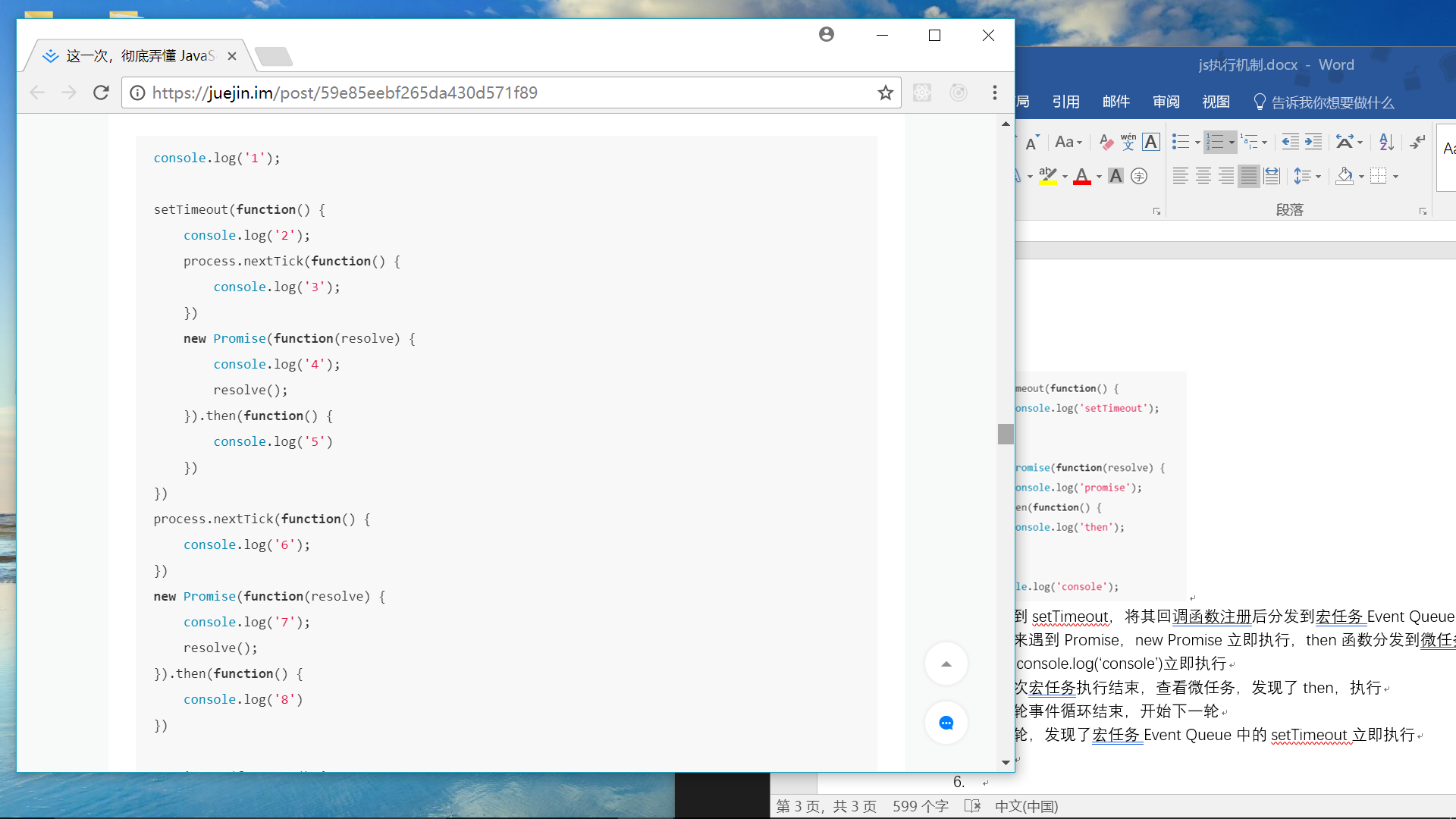
第一次宏任务执行结束，查看微任务，发现了then，执行

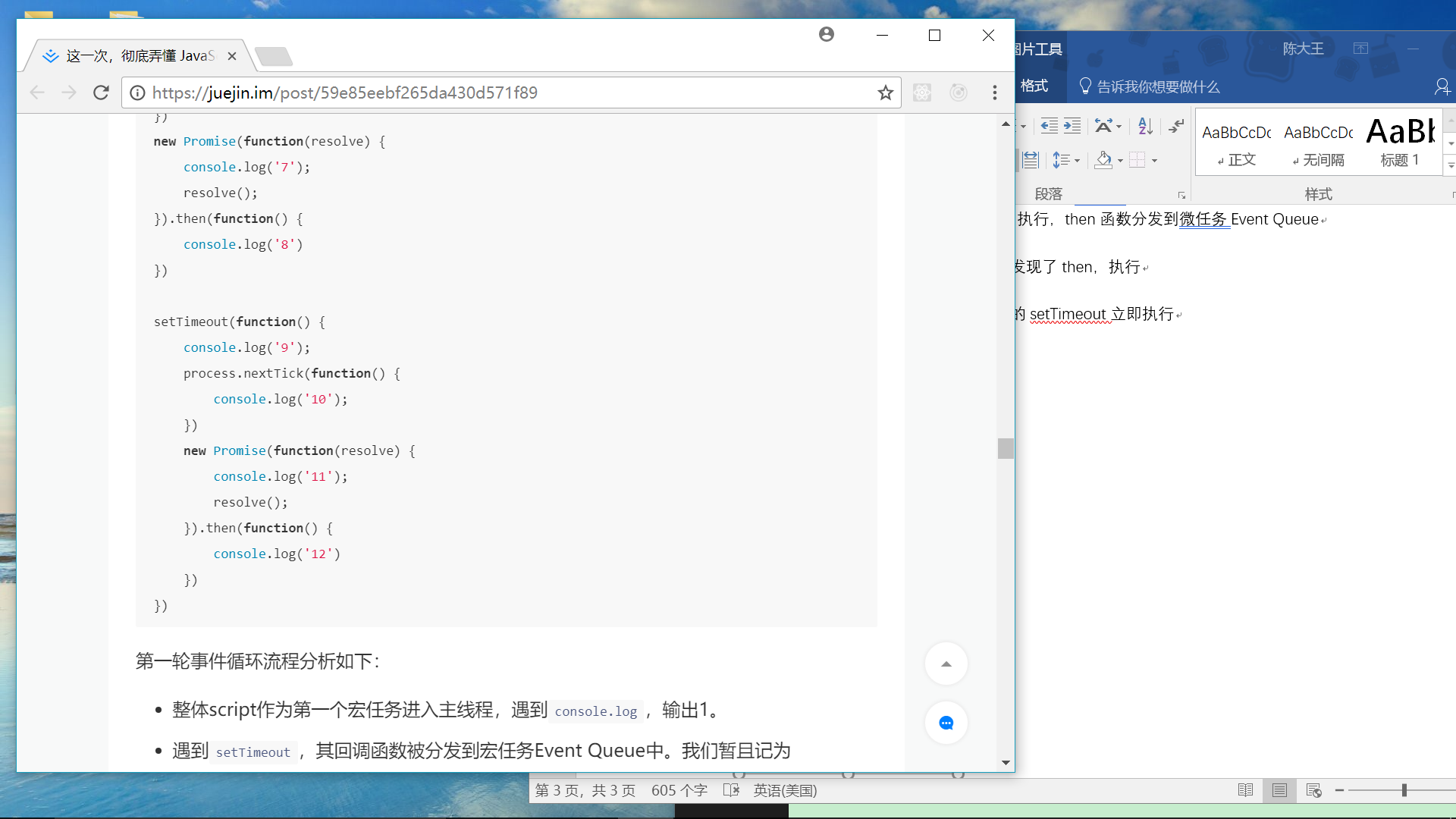
第一轮事件循环结束，开始下一轮

第二轮，发现了宏任务Event Queue中的setTimeout立即执行

结束

1. 复杂的例子





第一轮事件循环Event Loop

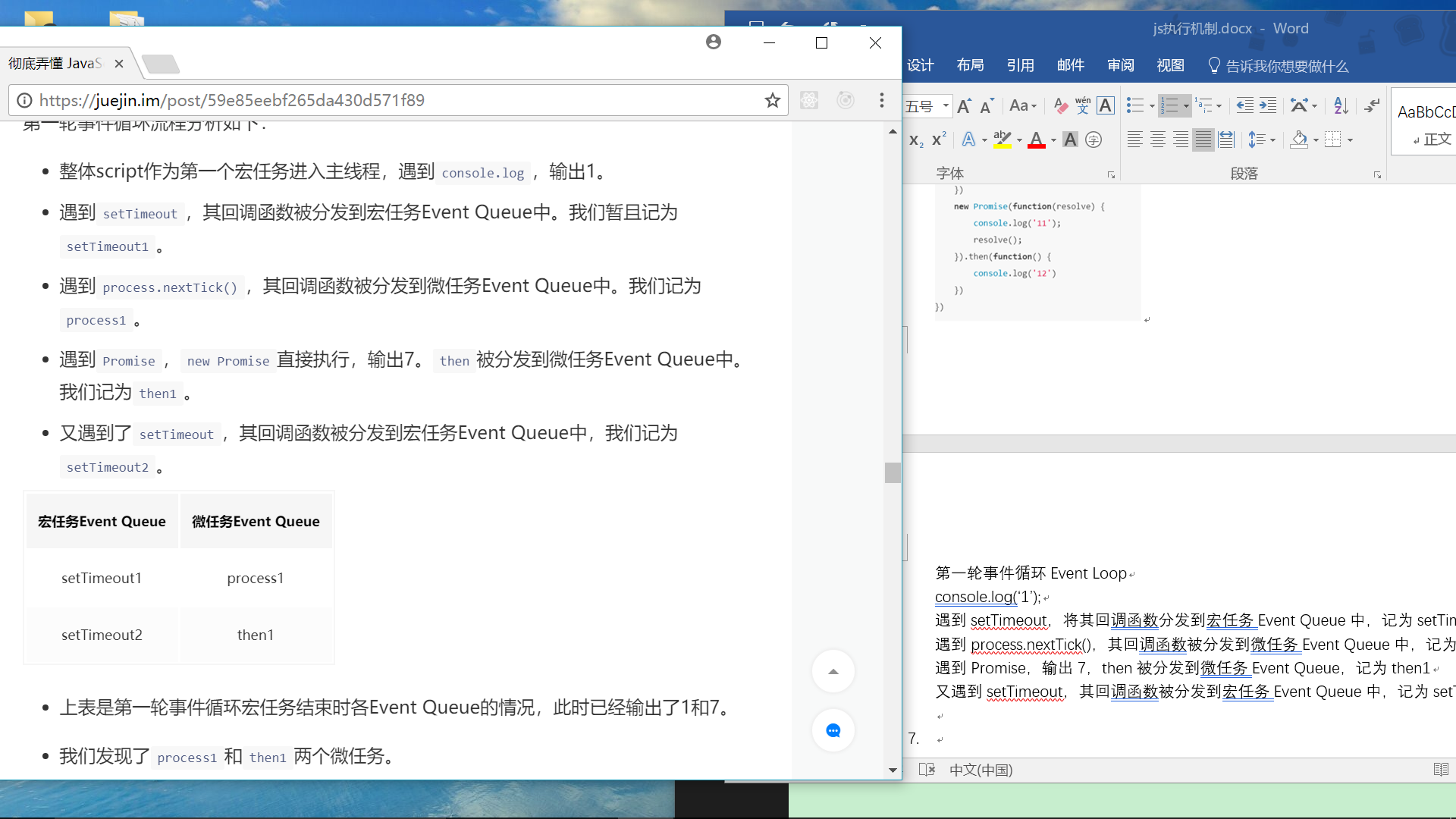
console.log(‘1’);

遇到setTimeout，将其回调函数分发到宏任务Event Queue中，记为setTimeout1

遇到process.nextTick()，其回调函数被分发到微任务Event Queue中，记为process1

遇到Promise，输出7，then被分发到微任务Event Queue，记为then1

又遇到setTimeout，其回调函数被分发到宏任务Event Queue中，记为setTimeout2



第一轮宏任务执行结束，开始执行微任务，输出6和8

第一轮事件循环输出1 7 6 8

第二轮事件循环Event Loop

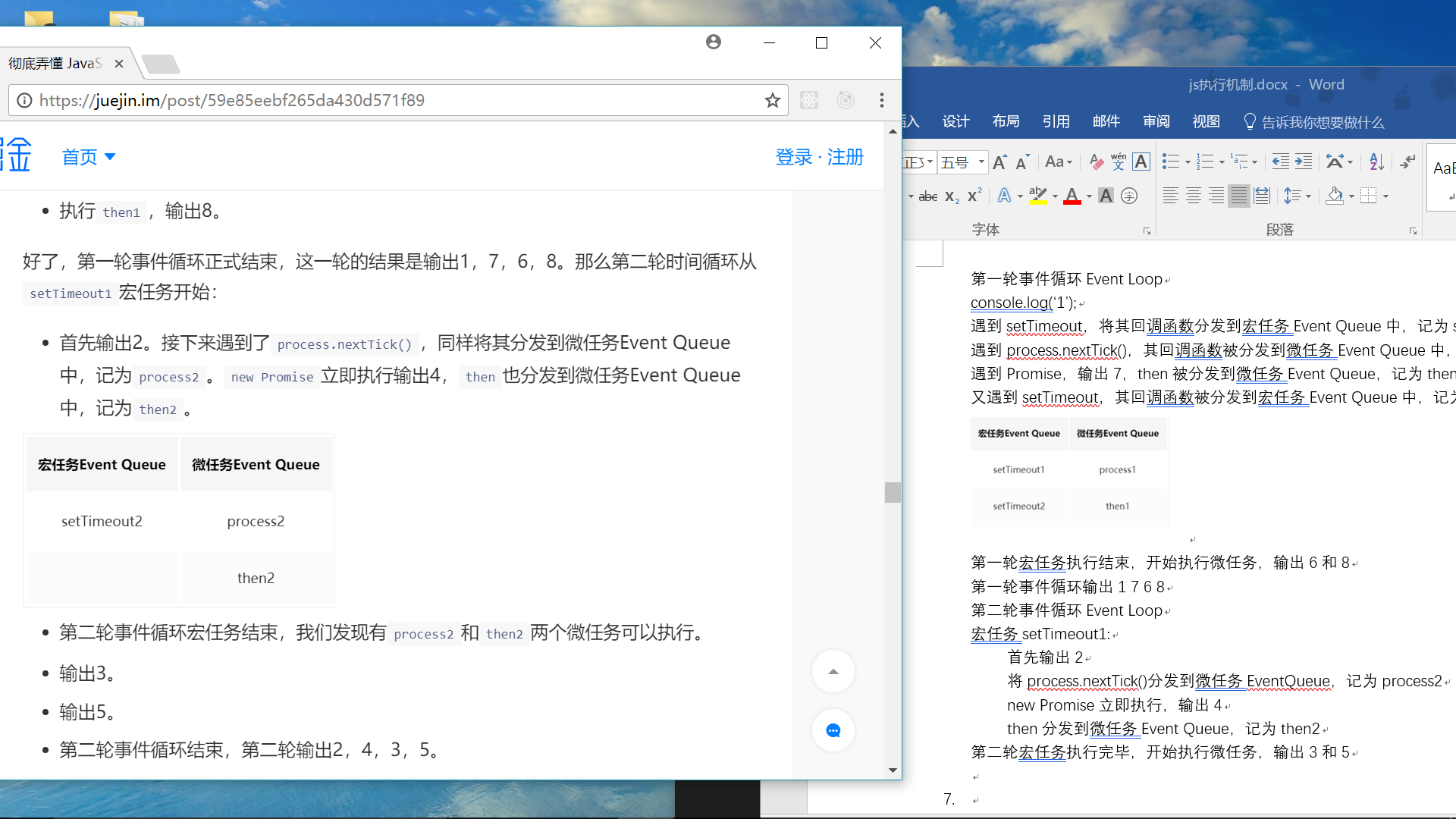
宏任务setTimeout1:

首先输出2

将process.nextTick()分发到微任务EventQueue，记为process2

new Promise立即执行，输出4

then分发到微任务Event Queue，记为then2



第二轮宏任务执行完毕，开始执行微任务，输出3和5

第二轮事件结束，输出2 4 3 5

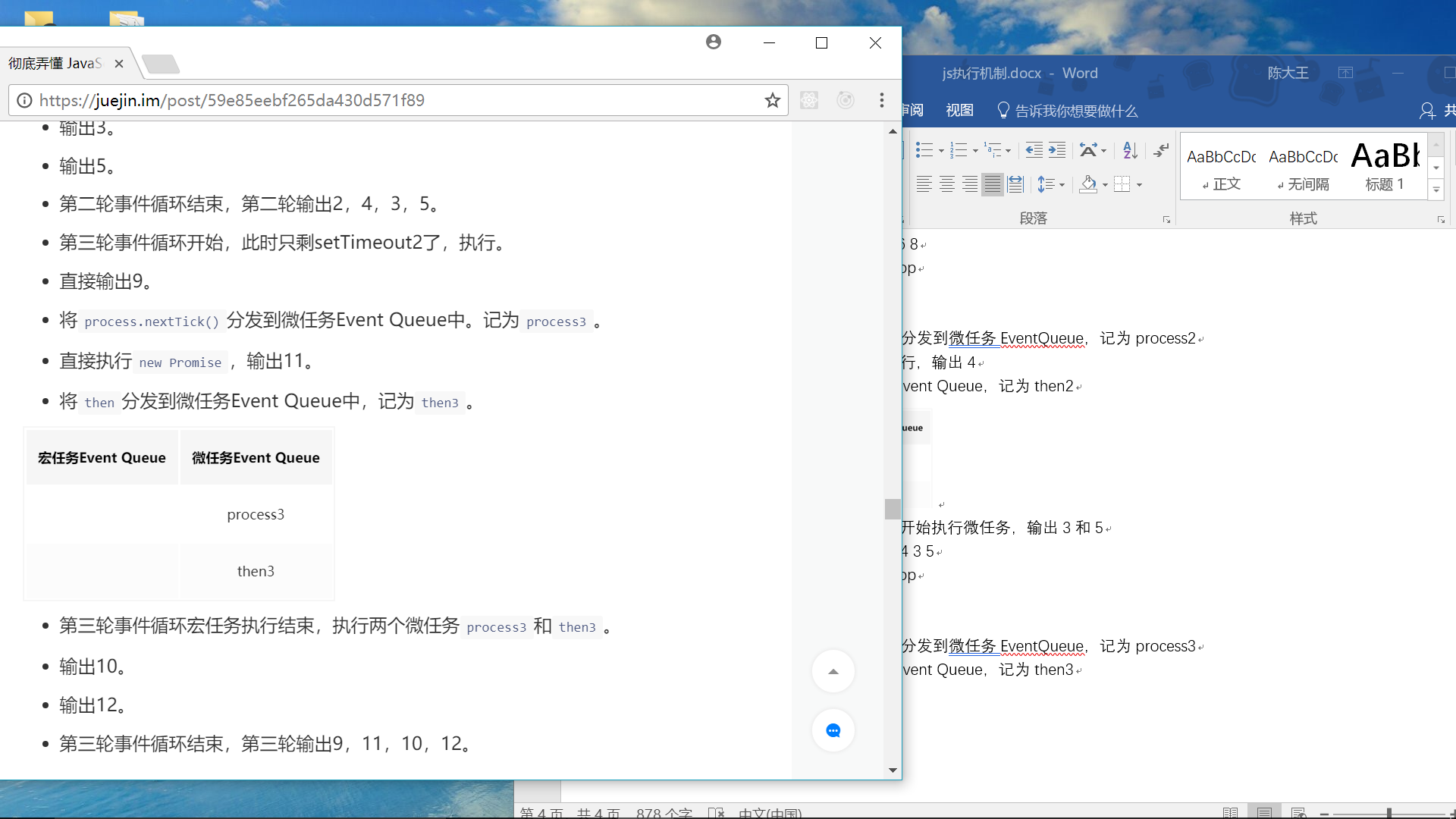
第三轮事件循环Event Loop

宏任务setTimeout2:

输出9

将process.nextTick()分发到微任务EventQueue，记为process3

then分发到微任务Event Queue，记为then3



第三轮宏任务执行完毕，开始执行微任务，输出10和12

第三轮事件结束，输出9 10 12

1. 总结

js是一门单线程语言

Event Loop是js的执行机制