JAVASCRIPT 运行机制


单线程


溯

J

V

t

为

览


脚

,

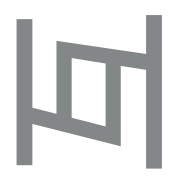
J

t

被

,

就



,

M

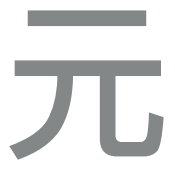
,

,

晰

多

,



,


1


D

,

而

却


浏

览

就


哪

为

"


制

,

提

V

S


性


他

的


大

,

t


始

就

7


路

,


5

提

W

W

,

t


脚

,


完

所

,


新

t

质

就

所


队

,

结

七

就


队

,

仅


队

子


呢

,

结

能

续

,

完

,


等

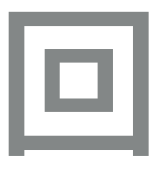
的

,

千

,


完



结

续

的


思

t

的


期

单线程

- 追溯到JavaScript的的诞生伊始。作为一门浏览器端脚本语言，JavaScript从被创造出的那一刻开始，就主要用于与用户互动，以及操作DOM，进行客户端校验，业务场景十分清晰简单。若以多线程的方式操作这些DOM，则可能出现操作冲突。假设有两个线程同时操作一个DOM元素，线程1要求浏览器删除DOM，而线程2却要求修改DOM样式，这时浏览器就无法决定采用哪个线程的操作。当然，我们可以为浏览器引入“锁”的机制来解决这个冲突，但这会大大提高javascript语言本身的复杂性。加上其他的一些原因，JavaScript 从诞生开始就选择了单线程这条道路
- 为了利用多核CPU的计算能力，HTML5提出Web Worker标准，允许JavaScript脚本创建多个线程，但是子线程完全受主线程控制，且不得操作DOM。所以，这个新标准并没有改变JavaScript单线程的本质
- 单线程就意味着，所有任务需要排队，前一个任务结束，才会执行后一个任务。如果前一个任务耗时很长，后一个任务就不得不一直等着。但是如果不是由于CPU处于忙碌状态而导致的排队，仅仅是由于IO设备导致的排队（此时CPU处于空闲状态呢,但是IO设备占用主线程），JS则必须等待IO的结果才能继续进行，这段时间CPU的空闲状态是浪费掉了。但是完全可以在这时主线程不管IO设备，挂起处于等待状态的任务，先运行处于后面的任务，这些耗时的任务完成后则以回调的方式执行相应处理，等IO操作结束了，再回过头，把挂起的任务继续执行。异步与回调的思想贯彻了整个JavaScript的生命周期

异步执行机制