**JavaScript引擎的执行机制**

首先，请牢记2点：

**1、JS是单线程语言**

**2、JS的Event Loop是JS的执行机制。深入了解JS的执行，就等于深入了解JS里的event loop**

**灵魂三问：JS为什么是单线程的？为什么需要异步？单线程又是如何实现异步的呢？**

**(1) JS为什么是单线程的？**

JS最初被设计用在浏览器中，那么想象一下，如果浏览器中的JS是多线程的。

场景描述：

那么现在有2个进程，process1 process2，由于是多进程的JS，所以他们对同一个dom，同时进行操作。 process1 删除了该dom，而process2 编辑了该dom，同时下达2个矛盾的命令，浏览器究竟该如何执行呢？这样想，JS为什么被设计成单线程应该就容易理解了吧。

**(2) JS为什么需要异步?**

场景描述：

如果JS中不存在异步，只能自上而下执行，如果上一行解析时间很长，那么下面的代码就会被阻塞。 对于用户而言，阻塞就意味着"卡死"，这样就导致了很差的用户体验。所以，JS中存在异步执行。

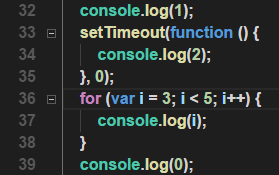
**(3) JS单线程又是如何实现异步的呢？**

既然JS是单线程的，只能在一条线程上执行，又是如何实现的异步呢？

是通过的事件循环(event loop)，理解了event loop机制，就理解了JS的执行机制。

**3.1.JS中的event loop(1)**

例1,观察它的执行顺序 执行结果



也就是说，setTimeout里的函数并没有立即执行，而是延迟了一段时间，满足一定条件后才去执行的，这类代码，我们叫异步代码。

所以，这里我们首先知道了JS里的一种分类方式，就是将任务分为：**同步任务**和**异步任务**。

**按照这种分类方式:JS的执行机制是：**

首先判断JS是同步还是异步，同步就进入主进程，异步就进入event table

异步任务在event table中注册函数，当满足触发条件后，被推入event queue

同步任务进入主线程后一直执行，直到主线程空闲时，才会去event queue中查看是否有可执行的异步任务，如果有就推入主进程中

以上三步循环执行，这就是event loop。

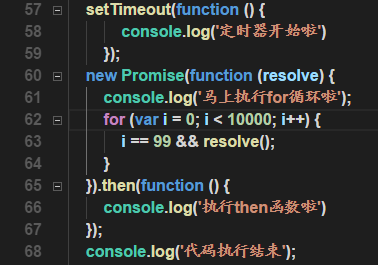
**所以上面的例子，你是否可以描述它的执行顺序了呢？**

console.log(1) 是同步任务，放入主线程里

setTimeout() 是异步任务，被放入event table， 0秒之后被推入event queue里

console.log(3 是同步任务，放到主线程里

当 1、 3在控制条被打印后，主线程去event queue(事件队列)里查看是否有可执行的函数，执行setTimeout里的函数。

**3.2.JS中的event loop(2)**

所以，上面关于event loop就是我对JS执行机制的理解，直到我遇到了下面这段代码。

例2：尝试按照，上文我们刚学到的JS执行机制去分析：

1.setTimeout 是异步任务，被放到event table;

2.new Promise 是同步任务，被放到主进程里，

直接执行打印 console.log('马上执行for循环啦');

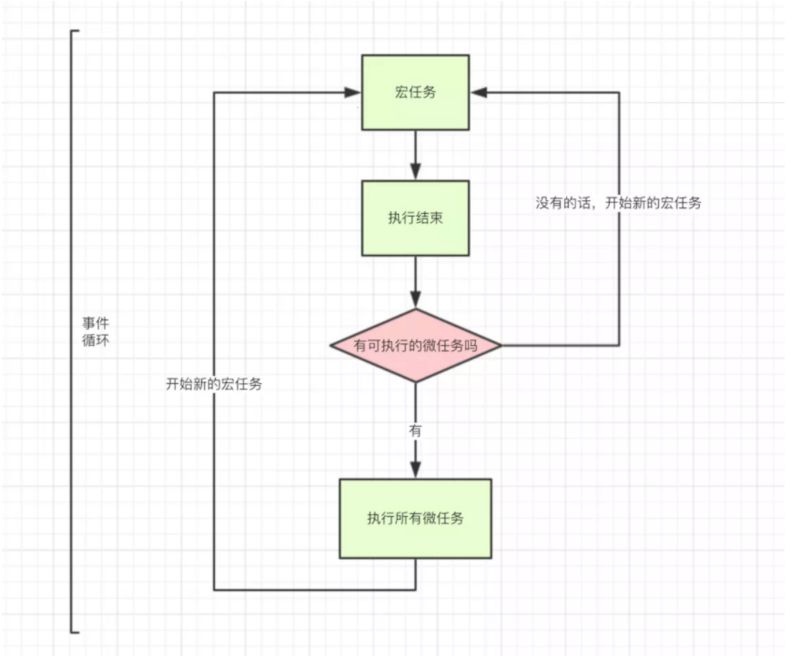
3..then 里的函数是异步任务，被放到event table;

4.console.log('代码执行结束') 是同步代码，被放到主进程里，直接执行.

所以，结果是：马上执行for循环啦---代码执行结束---定时器开始啦---执行then函数啦吗?

亲自执行后，结果居然不是这样，而是：

马上执行for循环啦---代码执行结束---执行then函数啦---定时器开始啦

**那么，难道是异步任务的执行顺序，不是前后顺序，而是另有规定？事实上，按照异步和同步的划分方式，并不准确。**

而准确的划分方式是：

macro-task(宏任务)：包括整体代码script，setTimeout，setInterval

micro-task(微任务)：Promise，process.nextTick

**按照这种分类方式，JS的执行机制是：**

执行一个宏任务，过程中如果遇到微任务，就将其放到微任务的“事件队列”里

当前宏任务执行完成后，会查看微任务的“事件队列”，并将里面全部的微任务依次执行完

**重复以上2步骤，结合event loop(1) event loop(2)，就是更为准确的JS执行机制了**

**尝试按照刚学的执行机制，去分析例2：**

首先执行script下的宏任务，遇到setTimeout,将其放到宏任务的“队列”里

遇到 new Promise直接执行，打印"马上执行for循环啦"

遇到then方法，是微任务，将其放到微任务的“队列”里。

打印 "代码执行结束"

本轮宏任务执行完毕，查看本轮的微任务，发现有一个then方法里的函数，打印"执行then函数啦"

到此,本轮的event loop 全部完成。

下一轮的循环里，先执行一个宏任务，发现宏任务的“队列”里有一个setTimeout里的函数,执行打印"定时器开始啦"

**所以最后的执行顺序是：马上执行for循环啦---代码执行结束---执行then函数啦---定时器开始啦**

**谈谈setTimeout**

这段setTimeout代码什么意思? 我们一般说: 3秒后,会执行setTimeout里的那个函数

setTimeout(function(){

   console.log('执行了')

},3000)

但是这种说并不严谨，准确的解释是：3秒后，setTimeout里的函数被会推入event queue，

而event queue(事件队列)里的任务，只有在主线程空闲时才会执行。

所以只有满足 (1)3秒后 (2)主线程空闲，同时满足时，才会3秒后执行该函数

如果主线程执行内容很多，执行时间超过3秒，比如执行了10秒，那么这个函数只能10秒后执行了。