# JS执行栈浅析

分享人：陈旭敏

时　间：2018年12月7日

# 目录

# 一. 基础概念

## 1.javascript是单线程语言

在浏览器中一个页面永远只有一个线程在执行js脚本代码

## 2. JS的Event Loop是JS的执行机制

javascript是单线程语言,但是代码解析却十分的快速，不会发生解析阻塞。

javascript是异步执行的，通过事件循环（Event Loop）的方式实现。

# 一. 基础概念

异步：程序中现在运行的部分和将来运行的部分之间的关系就是异步编程的核心。

例1：

```
function now(){
    return 21;
}

function later(){
    answer = answer * 2;
    console.log(answer);
}
var answer = now();
setTimeout(later,1000);
```

现在

```
function now(){
    return 21;
}
function later(){...}
var answer = now();
setTimeout(later,1000);
```

将来

```
answer = answer * 2;
console.log(answer);
```

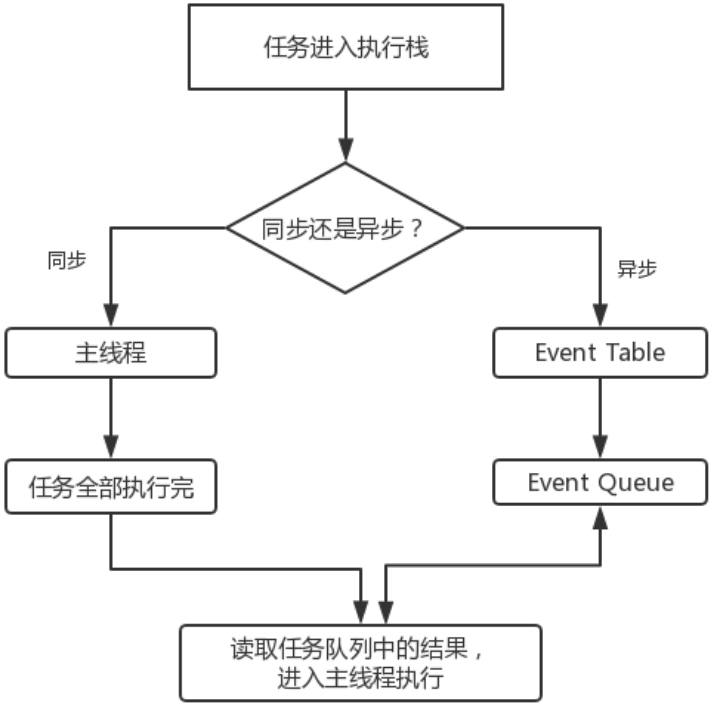
任何时候，只要将一段代码包装成一个函数，并指定它在响应某个事件(定时器、鼠标点击、Ajax响应等)时执行，你就是在代码中创建了一个将来执行的块，也由此在这个程序中引入了异步机制。

# 一. 基础概念

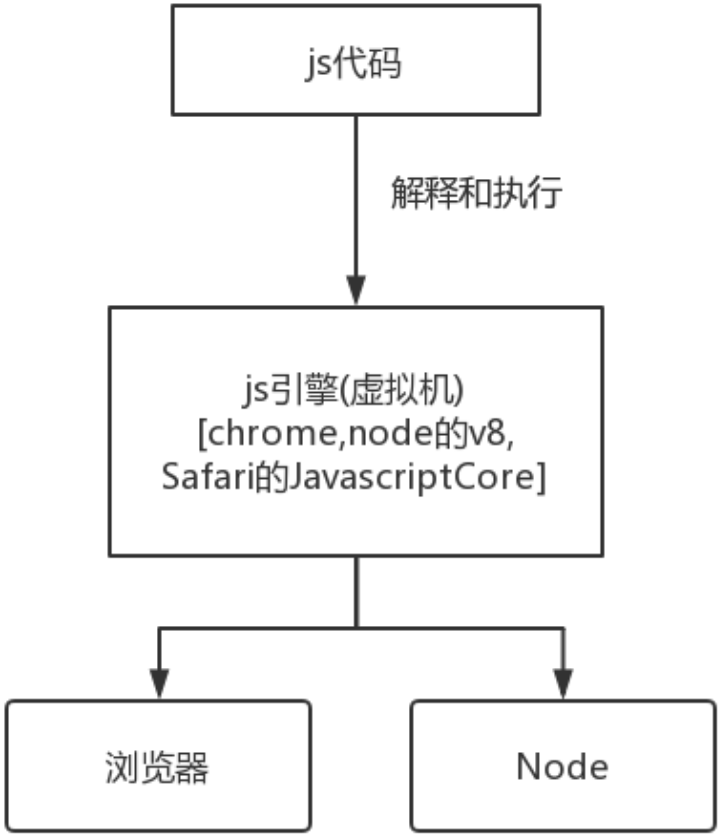
## 任务队列

任务队列是一个存储着待执行任务的队列，其中的任务严格按照时间先后顺序执行，排在队头的任务将会率先执行，而排在队尾的任务会最后执行。

## 异步运行机制

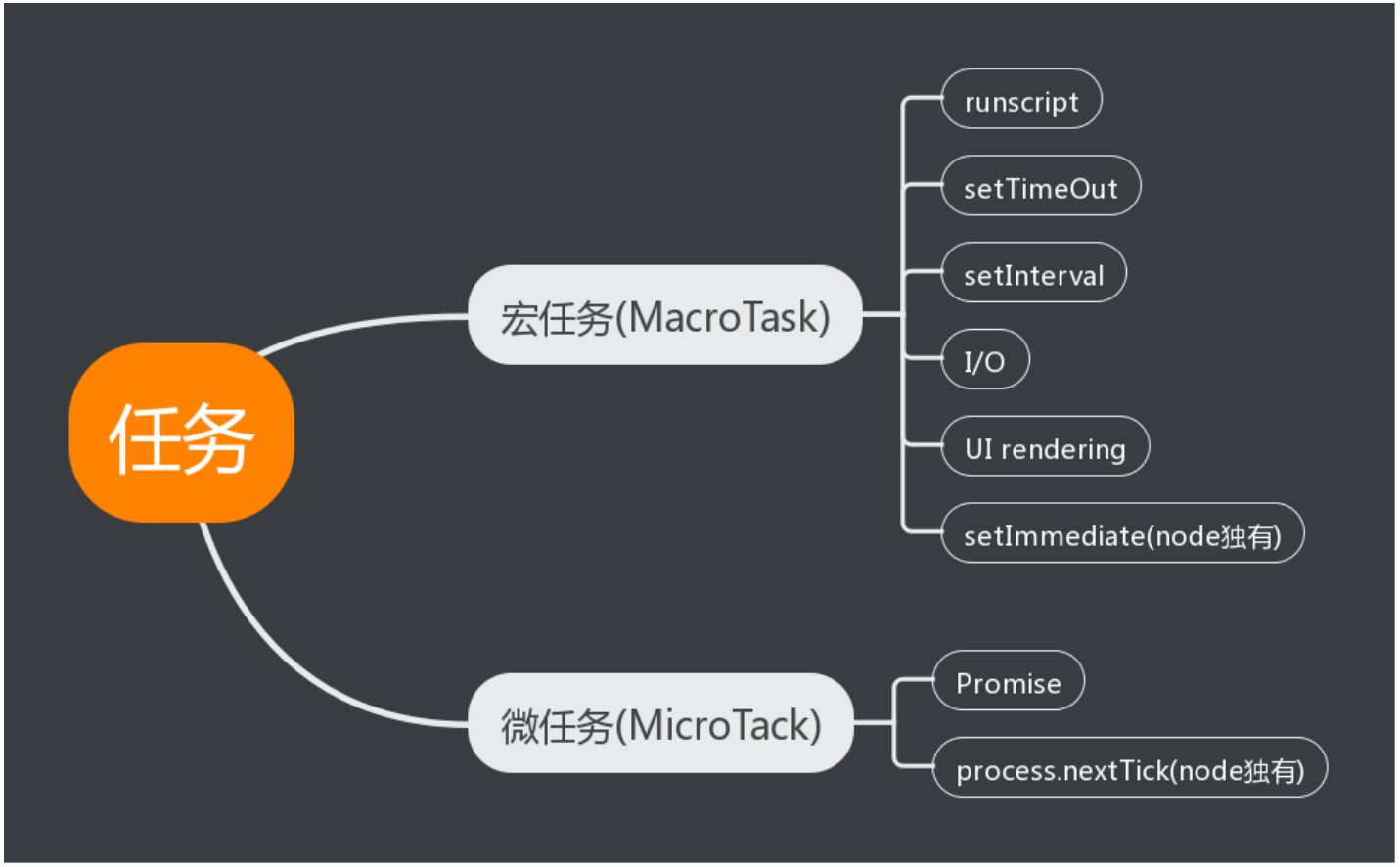


# 一. 基础概念


js代码
解释和执行
js引擎(虚拟机)
[chrome,node的v8,
Safari的JavascriptCore]
浏览器
Node

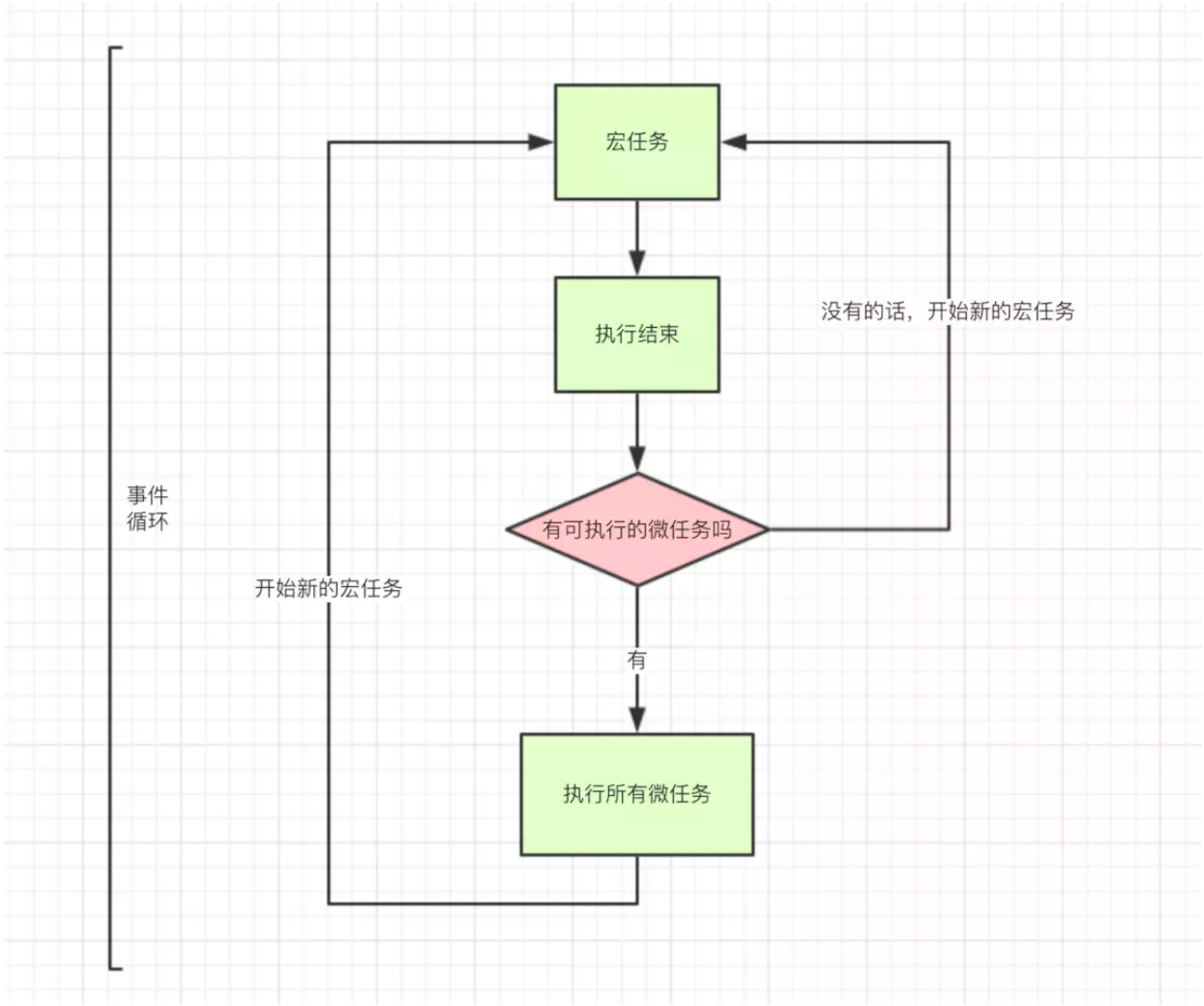
# 二. 浏览器中的事件循环

宏任务和微任务



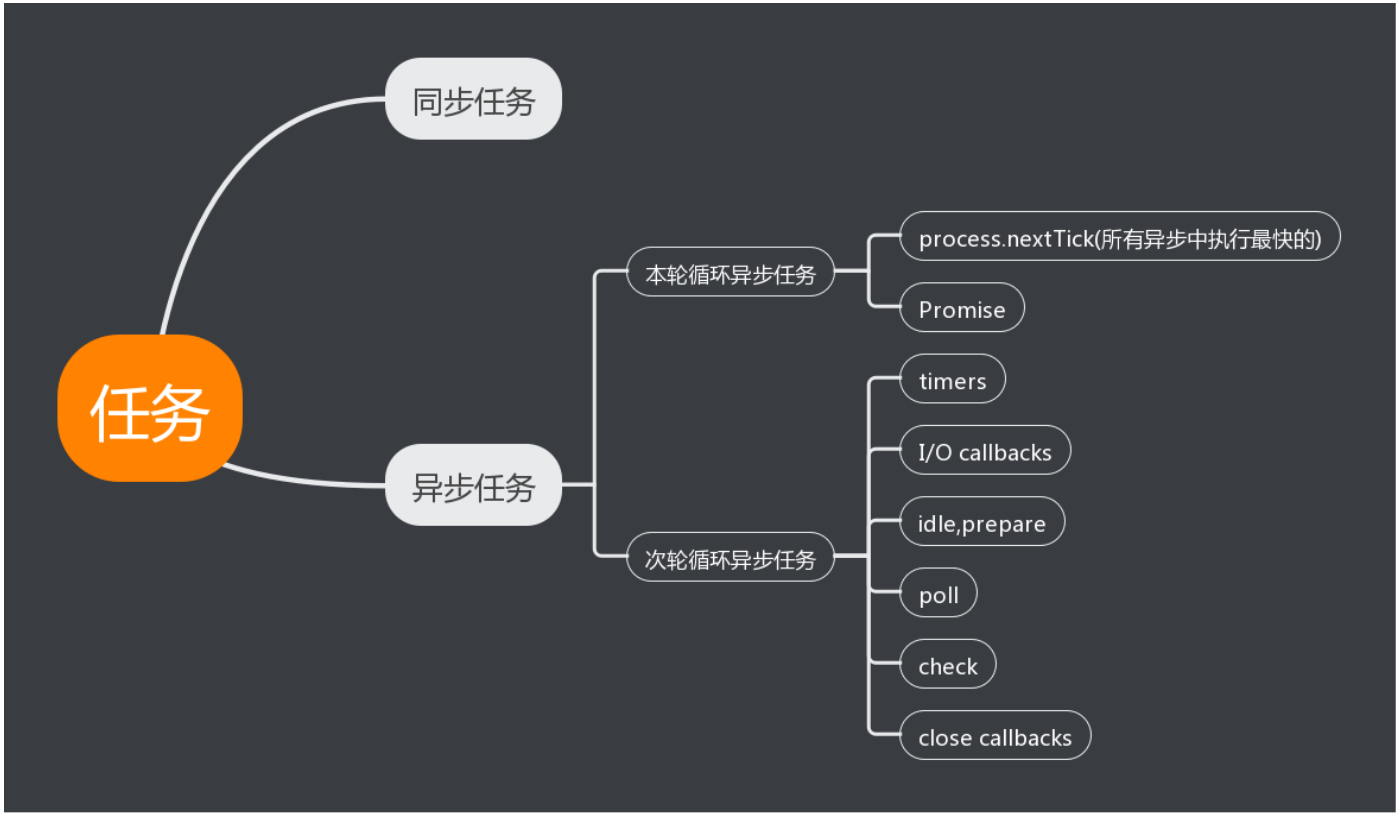
# 二. 浏览器中的事件循环

浏览器中的事件循环



# 三. Node中的事件循环

Node 的异步语法比浏览器更复杂，因为它可以跟内核对话，不得不弄了一个专门的库 libuv 做这件事。这个库负责各种回调函数的执行时间，毕竟异步任务最后还是要回到主线程，一个个排队执行。



# 三. Node中的事件循环

事件循环会无限次地执行，一轮又一轮。只有异步任务的回调函数队列清空了，才会停止执行。

每一轮的事件循环，分成六个阶段。这些阶段会**依次执行**。

1.Timers

这个是定时器阶段，处理setTimeout()和setInterval()的回调函数。进入这个阶段后，主线程会检查一下当前时间，是否满足定时器的条件。如果满足就执行回调函数，否则就离开这个阶段。

2.I/O callbacks

除了以下操作的回调函数，其他的回调函数都在这个阶段执行。

1)setTimeout()和setInterval()的回调函数

2)setImmediate()的回调函数

3)用于关闭请求的回调函数，比如socket.on('close', ...)

3.idle, prepare

该阶段只供 libuv 内部调用，这里可以忽略。

# 三. Node中的事件循环

4. Poll

这个阶段是轮询时间，用于等待还未返回的 I/O 事件，比如服务器的回应、用户移动鼠标等等。这个阶段的时间会比较长。如果没有其他异步任务要处理（比如到期的定时器），会一直停留在这个阶段，等待 I/O 请求返回结果。

5. check

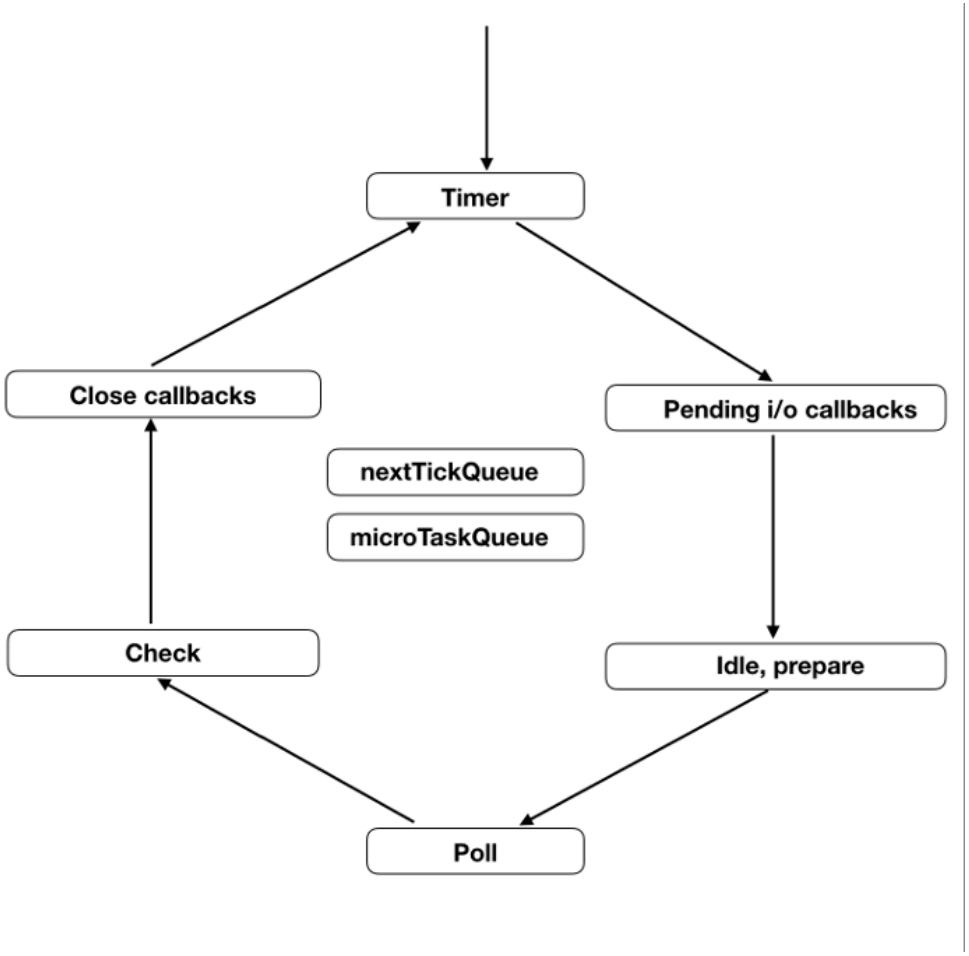
该阶段执行setImmediate()的回调函数。

6. close callbacks

该阶段执行关闭请求的回调函数，比如socket.on('close', ...)。

# 三. Node中的事件循环

每个阶段都有一个先进先出的回调函数队列。**只有一个阶段的回调函数队列清空了，该执行的回调函数都执行了，事件循环才会进入下一个阶段。**



# 四. 实践分析

例1. 最简单的同步异步示例

```
1
2   console.log('a');
3   setTimeout(() => {
4       console.log('b');
5   })
6   console.log('c');
```

执行结果： a c b

# 四. 实践分析

例2.

```
function sleep(d){
    for(var t = Date.now();Date.now() - t <= d;);
}
setTimeout(() => {
    console.log('a');
},3000);
console.log('b');
sleep(5000);
```

执行结果：b a(输出b 5s后输出)

# 四. 实践分析

例3.

```
1
2  console.log('a');
3  // set
4  setTimeout(() => {
5      console.log('b');
6  },0);
7
8  new Promise((resolve, reject) => {
9    console.log('c');
10   for (var i = 0; i < 10000; i++) {
11     i == 9999 && resolve();
12   }
13   console.log('d');
14   // pro
15 }).then(() => {
16   console.log('e');
17 });
18
19 console.log('f');
20
```

分析：

1.主线程开始执行

2.输出a

3.set进入宏任务队列

4.输出c

5.pro进入微任务队列

6.输出d

7.输出f

8.执行微任务队列pro，输出e,并将pro移除

9.执行宏任务队列中的第一个任务set，输出b，并将set移除

执行结果： a c d f e b

# 四. 实践分析

例4.

```
console.log('a')

// set1
setTimeout(() => {
  console.log('b')
  new Promise(resolve => {
    console.log('c')
    resolve();
    // pro1
  }).then(() => {
    console.log('d')
  })
})

new Promise(resolve => {
  console.log('e')
  resolve();
  // pro2
}).then(() => {
  console.log('f')
})

// set2
setTimeout(() => {
  console.log('g')
  new Promise(resolve => {
    console.log('h')
    resolve();
    // pro3
  }).then(() => {
    console.log('i')
  })
})
```

浏览器中执行分析：

1.主线程开始执行
2.输出a
3.set1进入宏任务队列
4.输出e
5.pro2进入微任务队列
6.set2进入宏任务队列
7.执行结束。此时宏任务队列中有set1,set2。微任务队列中有pro2。
8.遍历微任务队列。执行pro2，输出f，将pro2移除。
9.执行宏任务队列中set1，输出b,c，pro1进入微任务队列，set1移除
10.执行结束。此时宏任务队列中有set2。微任务队列中有pro1。
11.遍历微任务队列，执行pro1，输出d，将pro1移除
12.执行宏任务队列中set2，输出g,h,pro3进入微任务队列，set2移除
13.执行结束。此时宏任务队列中为空。微任务队列中有pro3。
14.遍历微任务队列，执行pro3，输出i，将pro3移除

执行结果：a e f b c d g h i

# 四. 实践分析

例4.

```
console.log('a')

// set1
setTimeout(() => {
  console.log('b')
  new Promise(resolve => {
    console.log('c')
    resolve();
    // pro1
  }).then(() => {
    console.log('d')
  })
})

new Promise(resolve => {
  console.log('e')
  resolve();
  // pro2
}).then(() => {
  console.log('f')
})

// set2
setTimeout(() => {
  console.log('g')
  new Promise(resolve => {
    console.log('h')
    resolve();
    // pro3
  }).then(() => {
    console.log('i')
  })
})
```

Node中执行分析：

1.主线程开始执行
2.输出a
3.set1进入timers队列
4.输出e
5.pro2进入微任务队列
6.set2进入timers队列
7.执行结束。此时timers队列中有set1,set2。微任务队列中有pro2。
8.遍历微任务队列。执行pro2，输出f，将pro2移除。
**9.遍历执行timers队列，执行set1，输出b,c，pro1进入微任务队列，set1移除。执行set2，输出g,h,pro3进入微任务队列。**
10.执行结束。此时timers队列中为空。微任务队列中有pro1，pro3。
11.遍历微任务队列，执行pro1，输出d，将pro1移除，执行pro3，输出i，将pro3移除

执行结果：a e f b c g h d i

# 五. 参考资料


1.JavaScript中的JS引擎的执行机制
https://zhuanlan.zhihu.com/p/33105489
2.从一道执行题，了解Node中JS执行机制
https://juejin.im/post/5b16a388f265da6e113f7a3d
3.JS引擎的执行机制
https://juejin.im/post/5a98c371518825556a71d397
4.一篇给小白看的 JavaScript 引擎指南
http://web.jobbole.com/84351/
5.微任务、宏任务与Event-Loop
https://juejin.im/post/5b73d7a6518825610072b42b
6.Node 定时器详解
http://www.ruanyifeng.com/blog/2018/02/node-event-loop.html
7.你不知道的JavaScript中卷

谢谢！