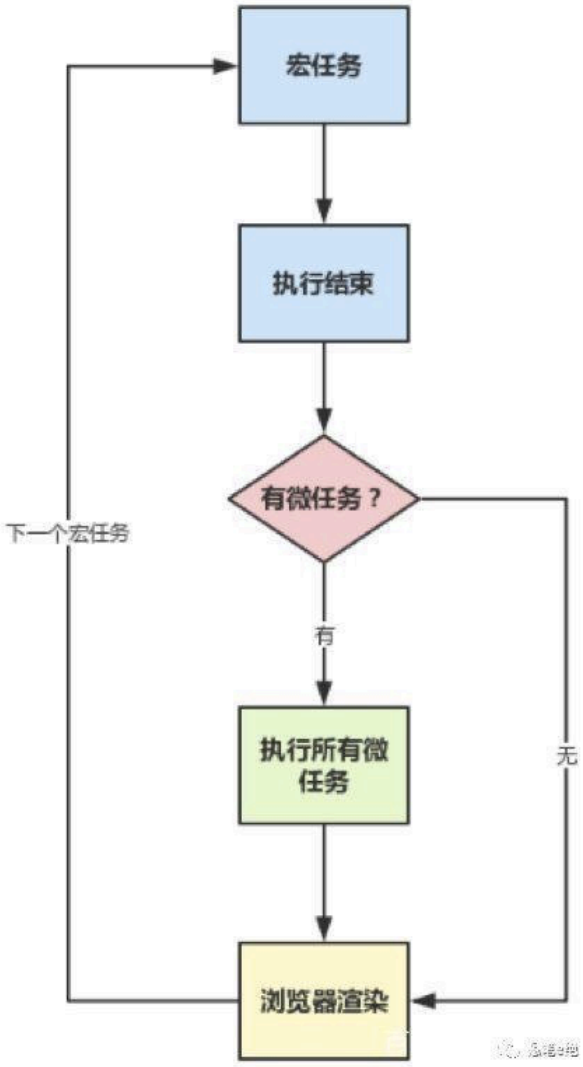
js事件循环机制

js的微任务和宏任务

- 执行一个宏任务（栈中没有就从事件队列中获取）
- 执行过程中如果遇到微任务，就将它添加到微任务的任务队列中
- 宏任务执行完毕后，立即执行当前微任务队列中的所有微任务（依次执行）
- 当前宏任务执行完毕，开始检查渲染，然后GUI线程接管渲染
- 渲染完毕后，JS线程继续接管，开始下一个宏任务（从事件队列中获取）



2.宏任务macrotask：
（事件队列中的每一个事件都是一个macrotask）
优先级：主代码块 > setImmediate > MessageChannel > setTimeout / setInterval
比如：setImmediate指定的回调函数，总是排在setTimeout前面
3.微任务包括：
优先级：process.nextTick > Promise > MutationObserver

实例

js 宏任务和微任务

```
setTimeout(() => {
    //执行后 回调一个宏事件
    console.log('内层宏事件3')
}, 0)
console.log('外层宏事件1');

new Promise((resolve) => {
    console.log('外层宏事件2');
    resolve()
 }).then(() => {
     console.log('微事件1');
 }).then(()=>{
     console.log('微事件2')
 })
```


```
//主线程直接执行
console.log('1');
//丢到宏事件队列中
setTimeout(function() {
    console.log('2');
    process.nextTick(function() {
        console.log('3');
    })
    new Promise(function(resolve) {
         console.log('4');
         resolve();
     }).then(function() {
         console.log('5')
     })
})
//微事件1
process.nextTick(function() {
    console.log('6');
})
//主线程直接执行
new Promise(function(resolve) {
    console.log('7');
    resolve();
}).then(function() {
```

```
    //微事件2
    console.log('8')
})
//丢到宏事件队列中
setTimeout(function() {
    console.log('9');
    process.nextTick(function() {
        console.log('10');
    })
    new Promise(function(resolve) {
        console.log('11');
        resolve();
    }).then(function() {
        console.log('12')
    })
})
```

多提一嘴async/await函数

因为，`async/await`本质上还是基于`Promise`的一些封装，而`Promise`是属于微任务的一种。所以在使用`await`关键字与`Promise.then`效果类似：

```
setTimeout(_ => console.log(4))

async function main() {
  console.log(1)
  await Promise.resolve()
  console.log(3)
}

main()

console.log(2)
// 1 2 3 4
```

async函数在await之前的代码都是同步执行的，可以理解为await之前的代码属于`new Promise`时传入的代码，await之后的所有代码都是在`Promise.then`中的回调

多个定时器，时间短先执行

```
setTimeout(()=>{
    console.log('定时器1');
}, 3000);

setTimeout(() => {
    console.log('定时器2');
}, 2000);

console.log('主线程');

for (let index = 0; index < 10000; index++) {
    console.log('循环');
}

console.log('循环之后');
```

```
13  setTimeout(()=>{
14      console.log('定时器1');
15  }, 3000);
16
17  setTimeout(() => {
18      console.log('定时器2');
19  }, 2000);
20
21  console.log('主线程');
22
23  for (let index = 0; index < 10000; index++) {
24      console.log('循环');
25  }
26
27  console.log('循环之后');
28
```

问题　输出　调试控制台　终端

```
循环
循环
循环
循环
循环
循环
循环
循环
循环
循环
循环
循环
循环之后
定时器2
定时器1
licai@licaideMacBook-Pro srm-sm-web %
```