

宏任务 macroTask 和微任务 microTask

- ◆ 什么是宏任务，什么是微任务
- ◆ event loop 和 DOM 渲染
- ◆ 微任务和宏任务的区别

慕课网

宏任务和微任务

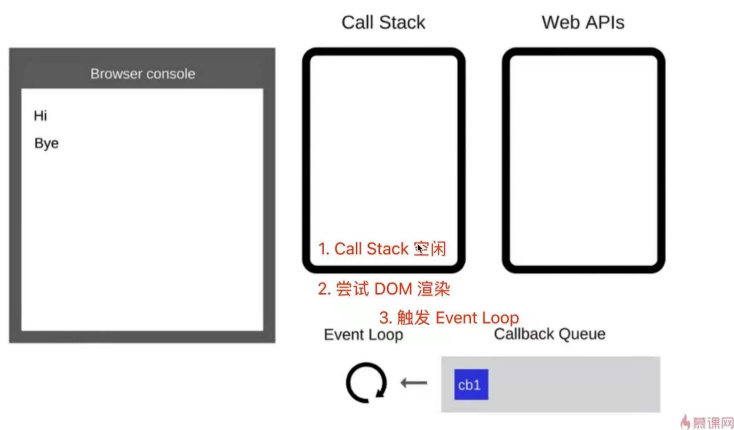
- ◆ 宏任务：setTimeout，setInterval，Ajax，DOM 事件
- ◆ 微任务：Promise async/await
- ◆ 微任务执行时机比宏任务要早（先记住）

慕课网

event loop 和 DOM 渲染

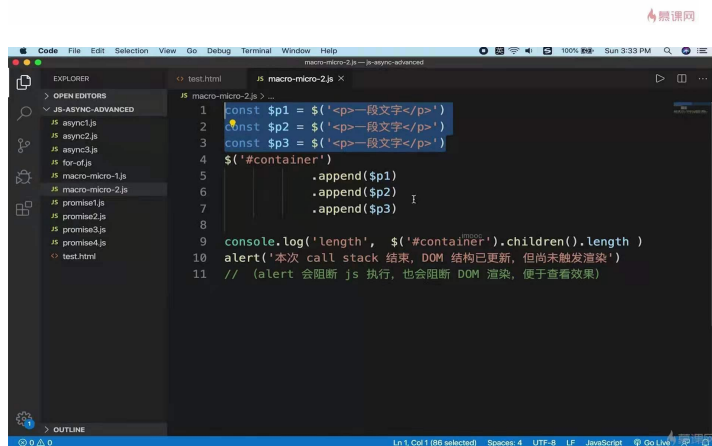
- ◆ 再次回归一遍 event loop 的过程
- ◆ JS 是单线程的，而且和 DOM 渲染共用一个线程
- ◆ JS 执行的时候，得留一些时机供 DOM 渲染

慕课网



event loop 和 DOM 渲染

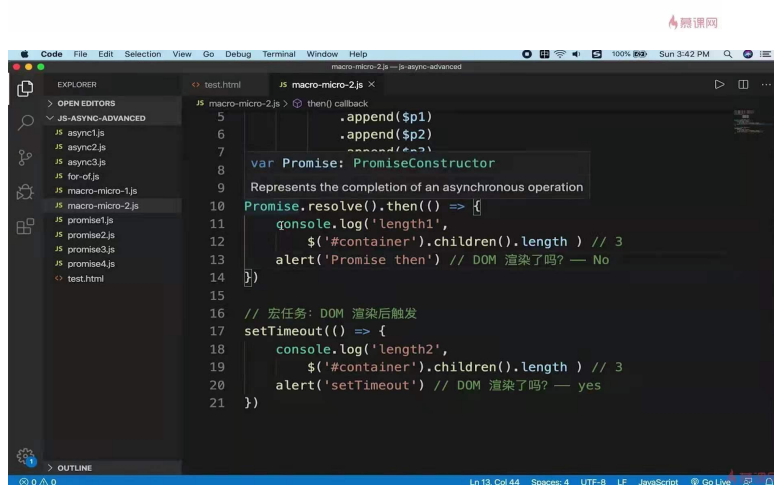
- ◆ 每次 Call Stack 清空（即每次轮询结束），即同步任务执行完
- ◆ 都是 DOM 重新渲染的机会，DOM 结构如有改变则重新渲染
- ◆ 然后再去触发下一次 Event Loop



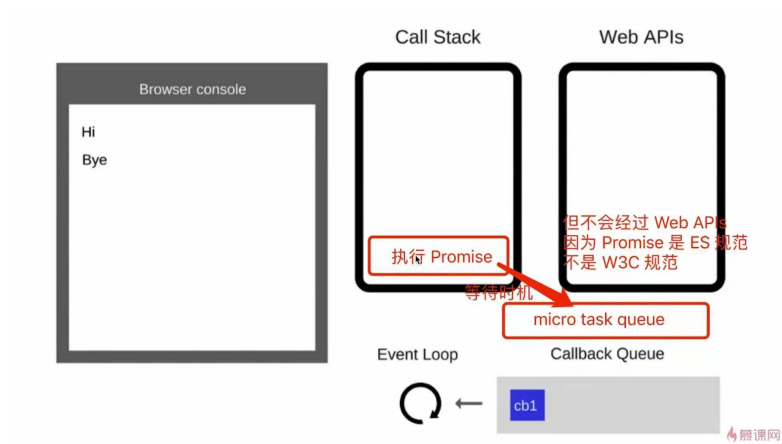
```
1 const $p1 = $('<p>一段文字</p>')
2 const $p2 = $('<p>一段文字</p>')
3 const $p3 = $('<p>一段文字</p>')
4 $('#container')
5     .append($p1)
6     .append($p2)
7     .append($p3)
8
9 console.log('length', $('#container').children().length)
10 alert('本次 call stack 结束, DOM 结构已更新, 但尚未触发渲染')
11 // (alert 会阻断 js 执行, 也会阻断 DOM 渲染, 便于查看效果)
```

微任务和宏任务的区别

- ◆ 宏任务：DOM 渲染后触发，如 setTimeout
- ◆ 微任务：DOM 渲染前触发，如 Promise
- ◆ 先演示现象，稍后再追究原理

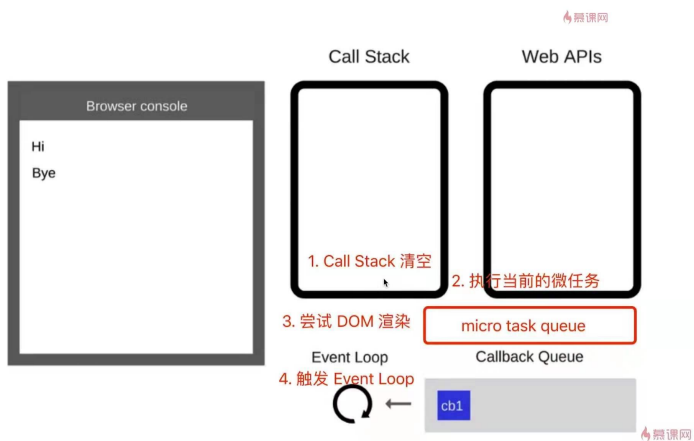


```
5     .append($p1)
6     .append($p2)
7     .append($p3)
8
9 var Promise: PromiseConstructor
10 Represents the completion of an asynchronous operation
11 Promise.resolve().then(() => {
12     console.log('length1',
13         $('#container').children().length ) // 3
14     alert('Promise then') // DOM 渲染了吗? — No
15 })
16
17 // 宏任务: DOM 渲染后触发
18 setTimeout(() => {
19     console.log('length2',
20         $('#container').children().length ) // 4
21     alert('setTimeout') // DOM 渲染了吗? — yes
22 })
```



为什么？

- ◆ 微任务是 ES6 语法规定的
- ◆ 宏任务是由浏览器规定的



微任务和宏任务 - 总结

- ◆ 宏任务有哪些？微任务有哪些？微任务触发时机更早
- ◆ 微任务、宏任务和 DOM 渲染的关系
- ◆ 微任务、宏任务和 DOM 渲染，在 event loop 的过程

什么是宏任务和微任务，两者区别

- ◆ 宏任务：setTimeout，setInterval，Ajax，DOM 事件
- ◆ 微任务：Promise，async/await
- ◆ 微任务执行时机比宏任务要早

描述 event loop 机制（可画图）

- ◆ 自行回顾 event loop 的过程
- ◆ 和 DOM 渲染的关系
- ◆ 微任务和宏任务在 event loop 过程中的不同处理

Promise 的三种状态，如何变化

- ◆ pending resolved rejected
- ◆ pending → resolved 或 pending → rejected
- ◆ 变化不可逆

场景题 - promise then 和 catch 的连接

```
// 第一题
Promise.resolve().then(() => {
  console.log(1)
}).catch(() => {
  console.log(2)
}).then(() => {
  console.log(3)
})
```

```
// 第二题
Promise.resolve().then(() => {
  console.log(1)
  throw new Error('error1')
}).catch(() => {
  console.log(2)
}).then(() => {
  console.log(3)
})
```

```
// 第三题
Promise.resolve().then(() => {
  console.log(1)
  throw new Error('error1')
}).catch(() => {
  console.log(2)
}).catch(() => { // 这里是 catch
  console.log(3)
})
```

场景题 - async/await 语法

```
async function fn() {  
  return 100  
}  
  
(async function () {  
  const a = fn() // ??  
  const b = await fn() // ??  
})();
```

```
(async function () {  
  console.log('start')  
  const a = await 100  
  console.log('a', a)  
  const b = await Promise.resolve(200)  
  console.log('b', b)  
  const c = await Promise.reject(300)  
  console.log('c', c)  
  console.log('end')  
})(); // 执行完毕，打印出那些内容?
```

场景题 - promise 和 setTimeout 的顺序

```
console.log(100)  
setTimeout(() => {  
  console.log(200)  
})  
Promise.resolve().then(() => {  
  console.log(300)  
})  
console.log(400)
```

场景题 - 外加 async/await 的顺序问题

```
async function async1 () {  
  console.log('async1 start')  
  await async2()  
  console.log('async1 end')  
}  
  
async function async2 () {  
  console.log('async2')  
}  
  
console.log('script start')  
  
setTimeout(function () {  
  console.log('setTimeout')  
, 0)
```

```
// 连接左侧代码，一起阅读  
async1()  
  
new Promise (function (resolve) {  
  console.log('promise1')  
  resolve()  
}).then (function () {  
  console.log('promise2')  
})  
  
console.log('script end')
```