黑马程序员™ | 传智播客旗下 高端IT教育品牌
www.itheima.com

ES6模块化与异步编程高级用法

黑马程序员 www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

目录 Contents

- ES6 模块化
- Promise
- async/await
- EventLoop
- 宏任务和微任务
- API 接口案例

黑马程序员
www.itheima.com

ES6 模块化

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

1. 回顾：node.js 中如何实现模块化

node.js 遵循了 CommonJS 的模块化规范。其中：

- 导入其它模块使用 require() 方法
- 模块对外共享成员使用 module.exports 对象

模块化的好处：

大家都遵守同样的模块化规范写代码，降低了沟通的成本，极大方便了各个模块之间的相互调用，利人利己。

黑马程序员 www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

2. 前端模块化规范的分类

在 ES6 模块化规范诞生之前，JavaScript 社区已经尝试并提出了 AMD、CMD、CommonJS 等模块化规范。

但是，这些由社区提出的模块化标准，还是存在一定的差异性与局限性、并不是浏览器与服务器通用的模块化标准，例如：

- AMD 和 CMD 适用于浏览器端的 Javascript 模块化
- CommonJS 适用于服务器端的 Javascript 模块化

太多的模块化规范给开发者增加了学习的难度与开发的成本。因此，大一统的 ES6 模块化规范诞生了！

黑马程序员 www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

3. 什么是 ES6 模块化规范

ES6 模块化规范是浏览器端与服务器端通用的模块化开发规范。它的出现极大的降低了前端开发者的模块化学习成本，开发者不需再额外学习 AMD、CMD 或 CommonJS 等模块化规范。

ES6 模块化规范中定义：

- 每个 js 文件都是一个独立的模块
- 导入其它模块成员使用 import 关键字
- 向外共享模块成员使用 export 关键字

黑马程序员 www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

4. 在 node.js 中体验 ES6 模块化

node.js 中默认仅支持 CommonJS 模块化规范，若想基于 node.js 体验与学习 ES6 的模块化语法，可以按照如下两个步骤进行配置：

① 确保安装了 v14.15.1 或更高版本的 node.js

② 在 package.json 的根节点中添加 "type": "module" 节点

黑马程序员 www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

5. ES6 模块化的基本语法

ES6 的模块化主要包含如下 3 种用法：

① 默认导出与默认导入

② 按需导出与按需导入

③ 直接导入并执行模块中的代码

黑马程序员
www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

5.1 默认导出

默认导出的语法： export default 默认导出的成员

```
let n1 = 10 // 定义模块私有成员 n1
let n2 = 20 // 定义模块私有成员 n2（外界访问不到 n2，因为它没有被共享出去）
function show() {} // 定义模块私有方法 show

export default { // 使用 export default 默认导出语法，向外共享 n1 和 show 两个成员
  n1,
  show
}
```

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

5.1 默认导入

默认导入的语法： import 接收名称 from '模块标识符'

```
// 从 01_m1.js 模块中导入 export default 向外共享的成员
// 并使用 m1 成员进行接收
import m1 from './01_m1.js'

// 打印输出的结果为：
// { n1: 10, show: [Function: show] }
console.log(m1)
```

黑马程序员 www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

5.1 默认导出的注意事项

每个模块中，只允许使用唯一的一次 export default，否则会报错！

```
let n1 = 10 // 定义模块私有成员 n1
let n2 = 20 // 定义模块私有成员 n2（外界访问不到 n2，因为它没有被共享出去）
function show() {} // 定义模块私有方法 show

export default { // 使用 export default 默认导出语法，向外共享 n1 和 show 两个成员
  n1,
  show
}

// SyntaxError: Identifier '.default' has already been declared
export default {
  n2
}
```

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

5.1 默认导入的注意事项

默认导入时的接收名称可以任意名称，只要是合法的成员名称即可：

```
// m1 是合法的名称
import m1 from './01_m1.js'

// 123m 不是合法的名称，因为成员名称不能以数字开头
import 123m from './01_m1.js'
```

黑马程序员 www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

5.2 按需导出

按需导出的语法： export 按需导出的成员

```
// 当前模块为 03_m2.js

// 向外按需导出变量 s1
export let s1 = 'aaa'
// 向外按需导出变量 s2
export let s2 = 'ccc'
// 向外按需导出方法 say
export function say() {}
```

黑马程序员 www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

5.2 按需导入

按需导入的语法： import { s1 } from '模块标识符'

```
// 导入模块成员
import { s1, s2, say } from './03_m2.js'

console.log(s1)  // 打印输出 aaa
console.log(s2)  // 打印输出 ccc
console.log(say) // 打印输出 [Function: say]
```

黑马程序员 www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

5.2 按需导出与按需导入的注意事项

① 每个模块中可以使用多次按需导出

② 按需导入的成员名称必须和按需导出的名称保持一致

③ 按需导入时，可以使用 as 关键字进行重命名

④ 按需导入可以和默认导入一起使用

黑马程序员
www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

5.3 直接导入并执行模块中的代码

如果只想单纯地执行某个模块中的代码，并不需要得到模块中向外共享的成员。此时，可以直接导入并执行模块代码，示例代码如下：

```
// 当前文件模块为 05_m3.js

// 在当前模块中执行一个 for 循环操作
for (let i = 0; i < 3; i++) {
  console.log(i)
}

// -------------------分割线-------------------------

// 直接导入并执行模块代码，不需要得到模块向外共享的成员
import './05_m3.js'
```

黑马程序员 www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

目录 Contents

- ES6 模块化
- Promise
- async/await
- EventLoop
- 宏任务和微任务
- API 接口案例

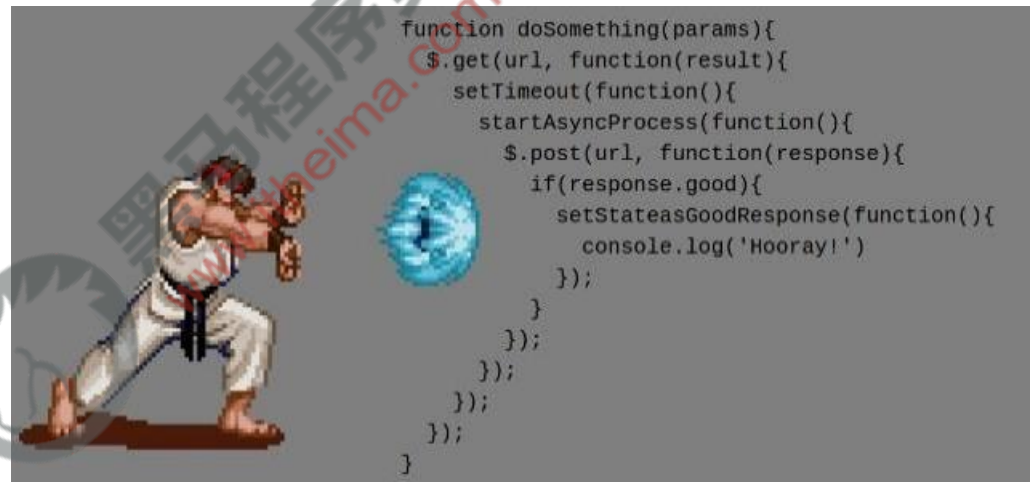
黑马程序员 www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

1. 回调地狱

多层回调函数的相互嵌套，就形成了回调地狱。示例代码如下：

```
1  setTimeout(() => { // 第 1 层回调函数
2    console.log('延时 1 秒后输出')
3
4    setTimeout(() => { // 第 2 层回调函数
5      console.log('再延时 2 秒后输出')
6
7      setTimeout(() => { // 第 3 层回调函数
8        console.log('再延时 3 秒后输出')
9      }, 3000)
10   }, 2000)
11 }, 1000)
```



回调地狱的缺点：

- 代码耦合性太强，牵一发而动全身，难以维护
- 大量冗余的代码相互嵌套，代码的可读性变差

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

1.1 如何解决回调地狱的问题

为了解决回调地狱的问题，ES6（ECMAScript 2015）中新增了 Promise 的概念。

黑马程序员
www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

1.2 Promise 的基本概念

① Promise 是一个构造函数

- 我们可以创建 Promise 的实例 const p = new Promise()
- new 出来的 Promise 实例对象，代表一个异步操作

② Promise.prototype 上包含一个 .then() 方法

- 每一次 new Promise() 构造函数得到的实例对象，
- 都可以通过原型链的方式访问到 .then() 方法，例如 p.then()

③ .then() 方法用来预先指定成功和失败的回调函数

- p.then(成功的回调函数，失败的回调函数)
- p.then(result => { }, error => { })
- 调用 .then() 方法时，成功的回调函数是必选的、失败的回调函数是可选的

黑马程序员 www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

2. 基于回调函数按顺序读取文件内容

```
// 读取文件 1.txt
fs.readFile('./files/1.txt', 'utf8', (err1, r1) => {
  if (err1) return console.log(err1.message) // 读取文件 1 失败
  console.log(r1) // 读取文件 1 成功
  // 读取文件 2.txt
  fs.readFile('./files/2.txt', 'utf8', (err2, r2) => {
    if (err2) return console.log(err2.message) // 读取文件 2 失败
    console.log(r2) // 读取文件 2 成功
    // 读取文件 3.txt
    fs.readFile('./files/3.txt', 'utf8', (err3, r3) => {
     if (err3) return console.log(err3.message) // 读取文件 3 失败
      console.log(r3)// 读取文件 3 成功
    })
  })
})
```

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

3. 基于 then-fs 读取文件内容

由于 node.js 官方提供的 fs 模块仅支持以回调函数的方式读取文件，不支持 Promise 的调用方式。因此，需要先运行如下的命令，安装 then-fs 这个第三方包，从而支持我们基于 Promise 的方式读取文件的内容：

```
npm install then-fs
```

黑马程序员 www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

3.1 then-fs 的基本使用

调用 then-fs 提供的 readFile() 方法，可以异步地读取文件的内容，它的返回值是 Promise 的实例对象。因此可以调用 .then() 方法为每个 Promise 异步操作指定成功和失败之后的回调函数。示例代码如下：

```
/**
 * 基于 Promise 的方式读取文件
 */
import thenFs from 'then-fs'
// 注意：.then() 中的失败回调是可选的，可以被省略
thenFs.readFile('./files/1.txt', 'utf8').then(r1 => { console.log(r1) }, err1 => { console.log(err1.message) })
thenFs.readFile('./files/2.txt', 'utf8').then(r2 => { console.log(r2) }, err2 => { console.log(err2.message) })
thenFs.readFile('./files/3.txt', 'utf8').then(r3 => { console.log(r3) }, err3 => { console.log(err3.message) })
```

注意：上述的代码无法保证文件的读取顺序，需要做进一步的改进！

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

3.2 .then() 方法的特性

如果上一个 .then() 方法中返回了一个新的 Promise 实例对象，则可以通过下一个 .then() 继续进行处理。通过 .then() 方法的链式调用，就解决了回调地狱的问题。

黑马程序员
www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

3.3 基于 Promise 按顺序读取文件的内容

Promise 支持链式调用，从而来解决回调地狱的问题。示例代码如下：

```
thenFs.readFile('./files/1.txt', 'utf8') // 1. 返回值是 Promise 的实例对象
  .then((r1) => { // 2. 通过 .then 为第一个 Promise 实例指定成功之后的回调函数
    console.log(r1)
    return thenFs.readFile('./files/2.txt', 'utf8') // 3. 在第一个 .then 中返回一个新的 Promise 实例对象
  })
  .then((r2) => {// 4. 继续调用 .then，为上一个 .then 的返回值（新的 Promise 实例）指定成功之后的回调函数
    console.log(r2)
    return thenFs.readFile('./files/3.txt', 'utf8') // 5. 在第二个 .then 中再返回一个新的 Promise 实例对象
  })
  .then((r3) => { // 6. 继续调用 .then，为上一个 .then 的返回值（新的 Promise 实例）指定成功之后的回调函数
    console.log(r3)
  })
```

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

3.4 通过 .catch 捕获错误

在 Promise 的链式操作中如果发生了错误，可以使用 Promise.prototype.catch 方法进行捕获和处理：

```
1 thenFs.readFile('./files/11.txt', 'utf8') // 文件不存在导致读取失败，后面的 3 个 .then 都不执行
2   .then(r1 => {
3     console.log(r1)
4     return thenFs.readFile('./files/2.txt', 'utf8')
5   })
6   .then(r2 => {
7     console.log(r2)
8     return thenFs.readFile('./files/3.txt', 'utf8')
9   })
10  .then(r3 => {
11    console.log(r3)
12  })
13  .catch(err => { // 捕获第 1 行发生的错误，并输出错误的消息
14    console.log(err.message)
15  })
```

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

3.4 通过 .catch 捕获错误

如果不希望前面的错误导致后续的 .then 无法正常执行，则可以将 .catch 的调用提前，示例代码如下：

```
1 thenFs.readFile('./files/11.txt', 'utf8')
2   .catch(err => {            // 捕获第 1 行发生的错误，并输出错误的消息
3     console.log(err.message) // 由于错误已被及时处理，不影响后续 .then 的正常执行
4   })
5   .then(r1 => {
6     console.log(r1) // 输出 undefined
7     return thenFs.readFile('./files/2.txt', 'utf8')
8   })
9   .then(r2 => {
10    console.log(r2) // 输出 222
11    return thenFs.readFile('./files/3.txt', 'utf8')
12  })
13  .then(r3 => {
14    console.log(r3) // 输出 333
15  })
```

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

3.5 Promise.all() 方法

Promise.all() 方法会发起并行的 Promise 异步操作，等所有的异步操作全部结束后才会执行下一步的 .then 操作（等待机制）。示例代码如下：

```
// 1. 定义一个数组，存放 3 个读文件的异步操作
const promiseArr = [
  thenFs.readFile('./files/11.txt', 'utf8'),
  thenFs.readFile('./files/2.txt', 'utf8'),
  thenFs.readFile('./files/3.txt', 'utf8'),
]
// 2. 将 Promise 的数组，作为 Promise.all() 的参数
Promise.all(promiseArr)
  .then(([r1, r2, r3]) => { // 2.1 所有文件读取成功（等待机制）
    console.log(r1, r2, r3)
  })
  .catch(err => { // 2.2 捕获 Promise 异步操作中的错误
    console.log(err.message)
  })
```

注意：数组中 Promise 实例的顺序，就是最终结果的顺序！

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

3.6 Promise.race() 方法

Promise.race() 方法会发起并行的 Promise 异步操作，只要任何一个异步操作完成，就立即执行下一步的 .then 操作（赛跑机制）。示例代码如下：

```
// 1. 定义一个数组，存放 3 个读文件的异步操作
const promiseArr = [
  thenFs.readFile('./files/1.txt', 'utf8'),
  thenFs.readFile('./files/2.txt', 'utf8'),
  thenFs.readFile('./files/3.txt', 'utf8'),
]
// 2. 将 Promise 的数组，作为 Promise.race() 的参数
Promise.race(promiseArr)
  .then((result) => { // 2.1 只要任何一个异步操作完成，就立即执行成功的回调函数（赛跑机制）
    console.log(result)
  })
  .catch(err => { // 2.2 捕获 Promise 异步操作中的错误
    console.log(err.message)
  })
```

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

4. 基于 Promise 封装读文件的方法

方法的封装要求：

① 方法的名称要定义为 getFile

② 方法接收一个形参 fpath，表示要读取的文件的路径

③ 方法的返回值为 Promise 实例对象

黑马程序员
www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

4.1 getFile 方法的基本定义

```
1 // 1. 方法的名称为 getFile
2 // 2. 方法接收一个形参 fpath，表示要读取的文件的路径
3 function getFile(fpath) {
4   // 3. 方法的返回值为 Promise 的实例对象
5   return new Promise()
6 }
```

注意：第 5 行代码中的 new Promise() 只是创建了一个形式上的异步操作。

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

4.2 创建具体的异步操作

如果想要创建具体的异步操作，则需要在 new Promise() 构造函数期间，传递一个 function 函数，将具体的异步操作定义到 function 函数内部。示例代码如下：

```
// 1. 方法的名称为 getFile
// 2. 方法接收一个形参 fpath，表示要读取的文件的路径
function getFile(fpath) {
  // 3. 方法的返回值为 Promise 的实例对象
  return new Promise(function() {
    // 4. 下面这行代码，表示这是一个具体的、读文件的异步操作
    fs.readFile(fpath, 'utf8', (err, dataStr) => {  })
  })
}
```

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

4.3 获取 .then 的两个实参

通过 .then() 指定的成功和失败的回调函数，可以在 function 的形参中进行接收，示例代码如下：

```
function getFile(fpath) {
  // resolve 形参是：调用 getFiles() 方法时，通过 .then 指定的“成功的”回调函数
  // reject  形参是：调用 getFiles() 方法时，通过 .then 指定的“失败的”回调函数
  return new Promise(function(resolve, reject) {
    fs.readFile(fpath, 'utf8', (err, dataStr) => { })
  })
}

// getFile 方法的调用过程：
getFile('./files/1.txt').then(成功的回调函数, 失败的回调函数)
```

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

4.4 调用 resolve 和 reject 回调函数

Promise 异步操作的结果，可以调用 resolve 或 reject 回调函数进行处理。示例代码如下：

```
function getFile(fpath) {
  // resolve 是“成功的”回调函数；reject 是“失败的”回调函数
  return new Promise(function(resolve, reject) {
    fs.readFile(fpath, 'utf8', (err, dataStr) => {
        if(err) return reject(err) // 如果读取失败，则调用“失败的回调函数”
        resolve(dataStr)           // 如果读取成功，则调用“成功的回调函数”
    })
  })
}

// getFile 方法的调用过程：
getFile('./files/1.txt').then(成功的回调函数, 失败的回调函数)
```

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

目录 Contents

- ES6 模块化
- Promise
- async/await
- EventLoop
- 宏任务和微任务
- API 接口案例

黑马程序员 www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

1. 什么是 async/await

async/await 是 ES8（ECMAScript 2017）引入的新语法，用来简化 Promise 异步操作。在 async/await 出现之前，开发者只能通过链式 .then() 的方式处理 Promise 异步操作。示例代码如下：

```
1 thenFs.readFile('./files/1.txt', 'utf8')
2   .then(r1 => {
3     console.log(r1)
4     return thenFs.readFile('./files/2.txt', 'utf8')
5   })
6   .then(r2 => {
7     console.log(r2)
8     return thenFs.readFile('./files/3.txt', 'utf8')
9   })
10  .then(r3 => {
11    console.log(r3)
12  })
```

.then 链式调用的优点：
解决了回调地狱的问题

.then 链式调用的缺点：
代码冗余、阅读性差、不易理解

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

2. async/await 的基本使用

使用 async/await 简化 Promise 异步操作的示例代码如下：

```
import thenFs from 'then-fs'

// 按照顺序读取文件 1，2，3 的内容
async function getAllFile() {
  const r1 = await thenFs.readFile('./files/1.txt', 'utf8')
  console.log(r1)
  const r2 = await thenFs.readFile('./files/2.txt', 'utf8')
  console.log(r2)
  const r3 = await thenFs.readFile('./files/3.txt', 'utf8')
  console.log(r3)
}

getAllFile()
```

黑马程序员 www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

3. async/await 的使用注意事项

① 如果在 function 中使用了 await，则 function 必须被 async 修饰

② 在 async 方法中，第一个 await 之前的代码会同步执行，await 之后的代码会异步执行

```
console.log('A')
async function getAllFile() {
  console.log('B')
  const r1 = await thenFs.readFile('./files/1.txt', 'utf8')
  const r2 = await thenFs.readFile('./files/2.txt', 'utf8')
  const r3 = await thenFs.readFile('./files/3.txt', 'utf8')
  console.log(r1, r2, r3)
  console.log('D')
}

getAllFile()
console.log('C')
```

```
// 最终输出的顺序
A
B
C
111 222 333
D
```

黑马程序员 www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

目录 Contents

- ES6 模块化
- Promise
- async/await
- EventLoop
- 宏任务和微任务
- API 接口案例

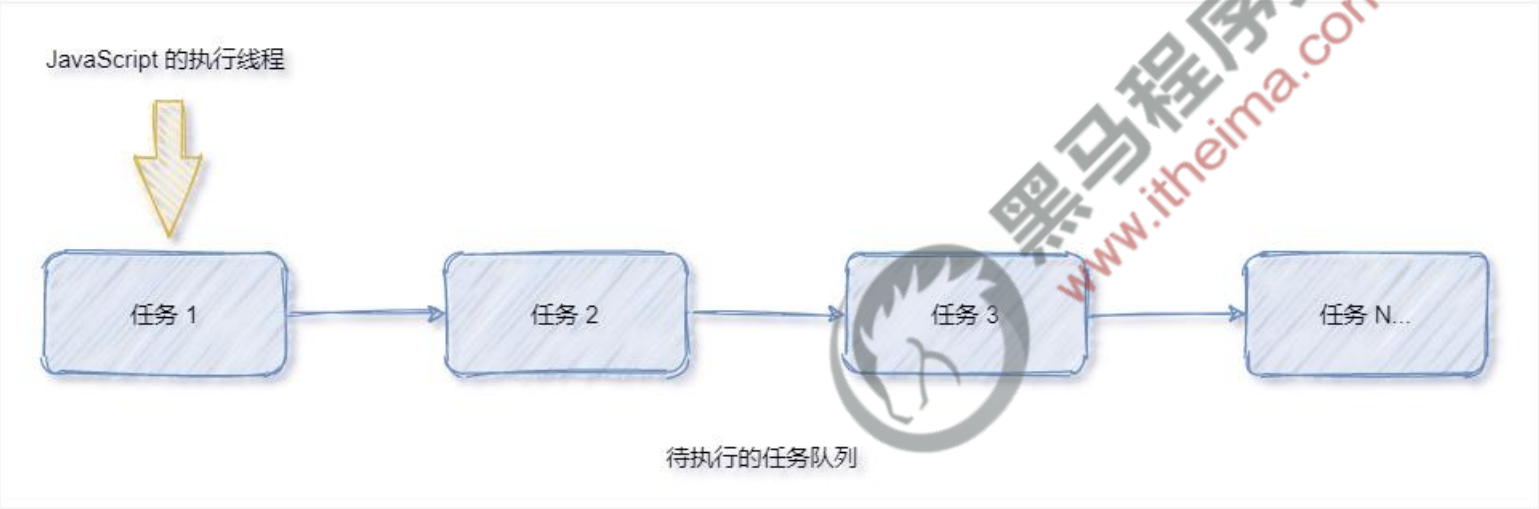
黑马程序员 www.itheima.com

EventLoop

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

1. JavaScript 是单线程的语言

JavaScript 是一门单线程执行的编程语言。也就是说，同一时间只能做一件事情。



单线程执行任务队列的问题：

如果前一个任务非常耗时，则后续的任务就不得不一直等待，从而导致程序假死的问题。

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

2. 同步任务和异步任务

为了防止某个耗时任务导致程序假死的问题，JavaScript 把待执行的任务分为了两类：

① 同步任务（synchronous）

- 又叫做非耗时任务，指的是在主线程上排队执行的那些任务
- 只有前一个任务执行完毕，才能执行后一个任务

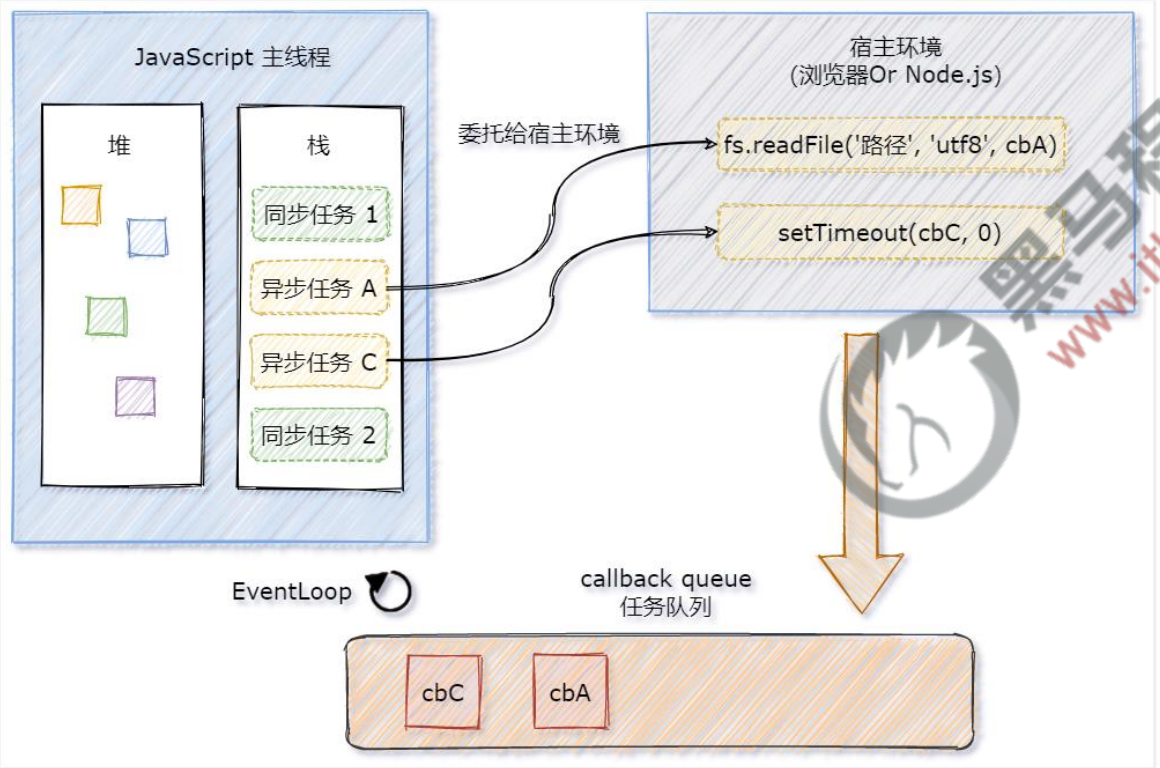
② 异步任务（asynchronous）

- 又叫做耗时任务，异步任务由 JavaScript 委托给宿主环境进行执行
- 当异步任务执行完成后，会通知 JavaScript 主线程执行异步任务的回调函数

黑马程序员 www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

3. 同步任务和异步任务的执行过程



① 同步任务由 JavaScript 主线程次序执行

② 异步任务委托给宿主环境执行

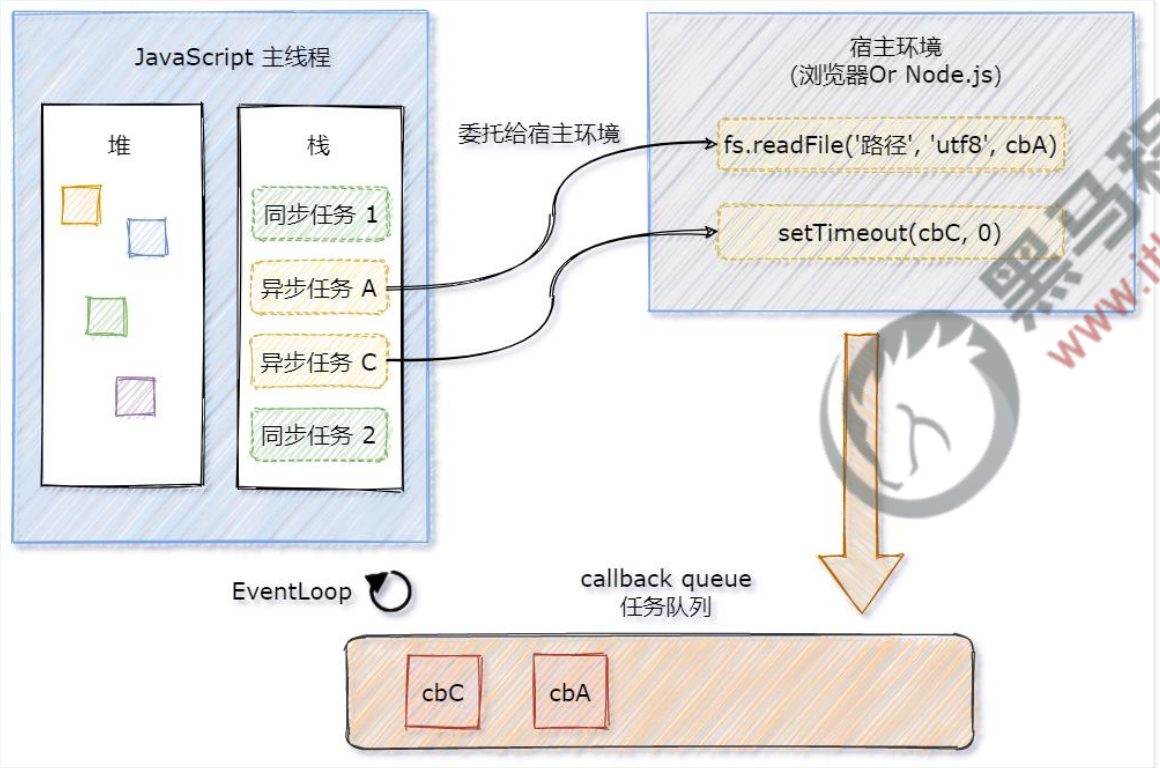
③ 已完成的异步任务对应的回调函数，会被加入到任务队列中等待执行

④ JavaScript 主线程的执行栈被清空后，会读取任务队列中的回调函数，次序执行

⑤ JavaScript 主线程不断重复上面的第 4 步

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

4. EventLoop 的基本概念



JavaScript 主线程从“任务队列”中读取异步任务的回调函数，放到执行栈中依次执行。这个过程是循环不断的，所以整个的这种运行机制又称为 EventLoop（事件循环）。

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

4. 结合 EventLoop 分析输出的顺序

```
1 import thenFs from 'then-fs'
2
3 console.log('A')
4 thenFs.readFile('./files/1.txt', 'utf8').then(dataStr => {
5   console.log('B')
6 })
7 setTimeout(() => {
8   console.log('C')
9 }, 0)
10 console.log('D')
```

正确的输出结果：ADCB。其中：

- A 和 D 属于同步任务。会根据代码的先后顺序依次被执行
- C 和 B 属于异步任务。它们的回调函数会被加入到任务队列中，等待主线程空闲时再执行

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

目录 Contents

- ES6 模块化
- Promise
- async/await
- EventLoop
- 宏任务和微任务
- API 接口案例

黑马程序员 www.itheima.com

宏任务和微任务

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

1. 什么是宏任务和微任务

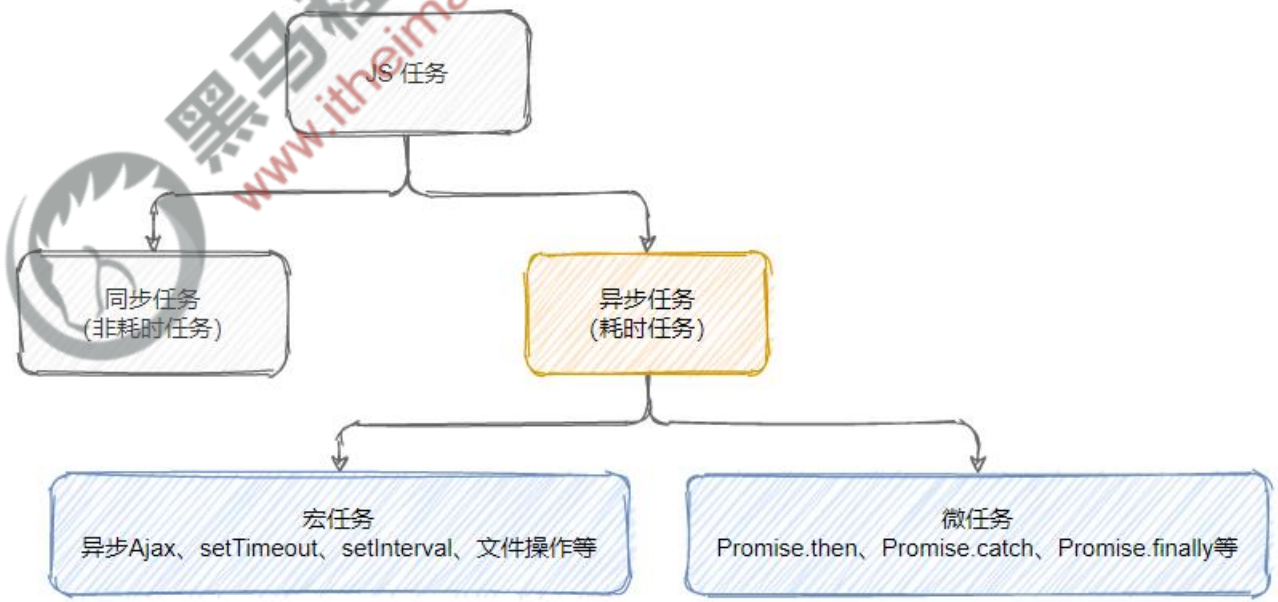
JavaScript 把异步任务又做了进一步的划分，异步任务又分为两类，分别是：

① 宏任务（macrotask）

- 异步 Ajax 请求、
- setTimeout、setInterval、
- 文件操作
- 其它宏任务

② 微任务（microtask）

- Promise.then、.catch 和 .finally
- process.nextTick
- 其它微任务

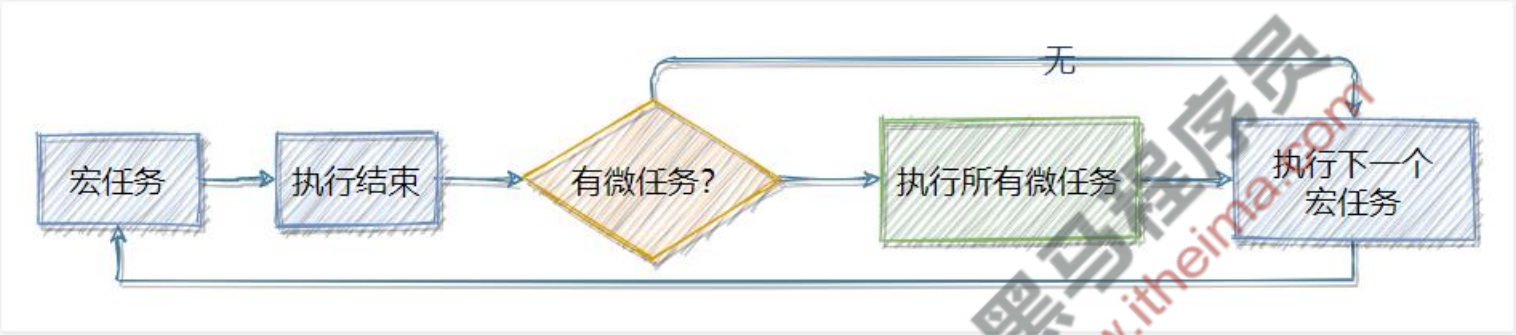


黑马程序员 www.itheima.com

宏任务和微任务

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

2. 宏任务和微任务的执行顺序



每一个宏任务执行完之后，都会检查是否存在待执行的微任务，

如果有，则执行完所有微任务之后，再继续执行下一个宏任务。

宏任务和微任务

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

3. 去银行办业务的场景

① 小云和小腾去银行办业务。首先，需要取号之后进行排队

- 宏任务队列

② 假设当前银行网点只有一个柜员，小云在办理存款业务时，小腾只能等待

- 单线程，宏任务按次序执行

③ 小云办完存款业务后，柜员询问他是否还想办理其它业务？

- 当前宏任务执行完，检查是否有微任务

④ 小云告诉柜员：想要买理财产品、再办个信用卡、最后再兑换点马年纪念币？

- 执行微任务，后续宏任务被推迟

⑤ 小云离开柜台后，柜员开始为小腾办理业务

- 所有微任务执行完毕，开始执行下一个宏任务

黑马程序员 www.itheima.com

宏任务和微任务

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

4. 分析以下代码输出的顺序

```
1 setTimeout(function () {
2   console.log('1')
3 })
4
5 new Promise(function (resolve) {
6   console.log('2')
7   resolve()
8 }).then(function () {
9   console.log('3')
10 })
11
12 console.log('4')
```

正确的输出顺序是：2431

分析：

① 先执行所有的同步任务

- 执行第 6 行、第 12 行代码

② 再执行微任务

- 执行第 9 行代码

③ 再执行下一个宏任务

- 执行第 2 行代码

黑马程序员 www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

5. 经典面试题

请分析以下代码输出的顺序（代码较长，截取成了左中右 3 个部分）：

```
console.log('1') //
setTimeout(function () {
  console.log('2') //
  new Promise(function (resolve) {
    console.log('3') //
    resolve()
  }).then(function () {
    console.log('4') //
  })
})
```

```
new Promise(function (resolve) {
  console.log('5') //
  resolve()
}).then(function () {
  console.log('6') //
})
```

```
setTimeout(function () {
  console.log('7') //
  new Promise(function (resolve) {
    console.log('8') //
    resolve()
  }).then(function () {
    console.log('9') //
  })
})
```

正确的输出顺序是：156234789

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

目录 Contents

- ES6 模块化
- Promise
- async/await
- EventLoop
- 宏任务和微任务
- API 接口案例

黑马程序员 www.itheima.com

API 接口案例

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

1. 案例需求

基于 MySQL 数据库 + Express 对外提供用户列表的 API 接口服务。用到的技术点如下：

- 第三方包 express 和 mysql2
- ES6 模块化
- Promise
- async/await

黑马程序员
www.itheima.com

API 接口案例

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

2. 主要的实现步骤

① 搭建项目的基本结构

② 创建基本的服务器

③ 创建 db 数据库操作模块

④ 创建 user_ctrl 业务模块

⑤ 创建 user_router 路由模块

黑马程序员
www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

3. 搭建项目的基本结构

① 启用 ES6 模块化支持

- 在 package.json 中声明 "type": "module"

② 安装第三方依赖包

- 运行 npm install express@4.17.1 mysql2@2.2.5

黑马程序员
www.itheima.com

API 接口案例

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

4. 创建基本的服务器

```
// 使用 ES6 的默认导入语法
import express from 'express'
const app = express()

app.listen(80, () => {
  console.log('server running at http://127.0.0.1')
})
```

黑马程序员 www.itheima.com

API 接口案例

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

5. 创建 db 数据库操作模块

```
import mysql from 'mysql2'

const pool = mysql.createPool({
  host: '127.0.0.1',
  port: 3306,
  database: 'my_db_01', // 请填写要操作的数据库的名称
  user: 'root', // 请填写登录数据库的用户名
  password: 'admin123', // 请填写登录数据库的密码
})

// 默认导出一个支持 Promise API 的 pool
export default pool.promise()
```

黑马程序员 www.itheima.com

API 接口案例

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

6. 创建 user_ctrl 模块

```
import db from '../db/index.js'

// 获取所有用户的列表数据
export async function getAllUser(req, res) {
  // db.query() 函数的返回值是 Promise 的实例对象。因此，可以使用 async/await 进行简化
  const [rows] = await db.query('select id, username, nickname from ev_users')
  res.send({
    status: 0,
    message: '获取用户列表数据成功！',
    data: rows,
  })
}
```

API 接口案例

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

7. 创建 user_router 模块

```
import express from 'express'
// 从 user_ctrl.js 模块中按需导入 getAllUser 函数
import { getAllUser } from '../controller/user_ctrl.js'

// 创建路由对象
const router = new express.Router()
// 挂载路由规则
router.get('/user', getAllUser)

// 使用 ES6 的默认导出语法，将路由对象共享出去
export default router
```

API 接口案例

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

8. 导入并挂载路由模块

```
import express from 'express'
// 1. 使用默认导入语法，导入路由对象
import userRouter from './router/user_router.js'
const app = express()

// 2. 挂载用户路由模块
app.use('/api', userRouter)

app.listen(80, () => {
  console.log('server running at http://127.0.0.1')
})
```

黑马程序员 www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

9. 使用 try...catch 捕获异常

```
1 export async function getAllUser(req, res) {
2   // 使用 try...catch 捕获 Promise 异步任务中产生的异常错误，并在 catch 块中进行处理
3   try {
4     // ev_users 表中没有 xxx 字段，所以此 SQL 语句会“执行异常”
5     const [rows] = await db.query('select id, username, nickname, xxx from ev_users')
6     res.send({ status: 0, message: '获取用户列表数据成功！', data: rows })
7   } catch (e) {
8     res.send({ status: 1, message: '获取用户列表数据失败！', desc: e.message })
9   }
10 }
```

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌

总结

① 能够知道如何使用 ES6 的模块化语法

- 默认导出与默认导入、按需导出与按需导入

② 能够知道如何使用 Promise 解决回调地狱问题

- promise.then()、promise.catch()

③ 能够使用 async/await 简化 Promise 的调用

- 方法中用到了 await，则方法需要被 async 修饰

④ 能够说出什么是 EventLoop

- EventLoop 示意图

⑤ 能够说出宏任务和微任务的执行顺序

- 在执行下一个宏任务之前，先检查是否有待执行的微任务

黑马程序员 www.itheima.com

黑马程序员
www.itheima.com
传智播客旗下高端IT教育品牌