AJAX进阶

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

目录

Contents

同步代码和异步代码

同步代码：

我们应该注意的是，实际上浏览器是按照我们书写代码的顺序一行一行地执行程序的。浏览器会等待代码的解析和工作，在上一行完成后才会执行下一行。这样做是很有必要的，因为每一行新的代码都是建立在前面代码的基础之上的。

这也使得它成为一个**同步程序**。

异步代码：

异步编程技术使你的程序可以在执行一个可能长期运行的任务的同时继续对其他事件做出反应而不必等待任务完成。与此同时，你的程序也将在任务完成后显示结果。

同步代码：逐行执行，需原地等待结果后，才继续向下执行

异步代码：调用后耗时，不阻塞代码继续执行（不必原地等待），在将来完成后触发一个回调函数

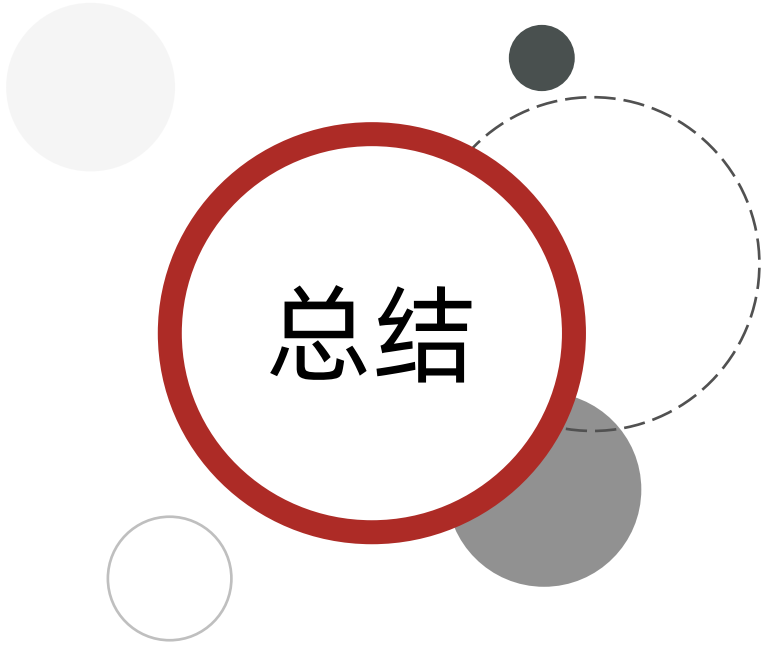
同步和异步

例子：回答打印数字的顺序是什么？

```
const result = 0 + 1
console.log(result)
setTimeout(() => {
  console.log(2)
}, 2000)
document.querySelector('.btn').addEventListener('click', () => {
  console.log(3)
})
document.body.style.backgroundColor = 'pink'
console.log(4)
```

打印结果：1,4,2
点击按钮一次就打印一次 3

异步代码接收结果：使用回调函数



1. 什么是同步代码？
   - 逐行执行，原地等待结果后，才继续向下执行
2. 什么是异步代码？
   - 调用后耗时，不阻塞代码执行，将来完成后触发回调函数
3. JS 中有哪些异步代码？
   - setTimeout / setInterval
   - 事件
   - AJAX
4. 异步代码如何接收结果？
   - 依靠回调函数来接收

目录

Contents

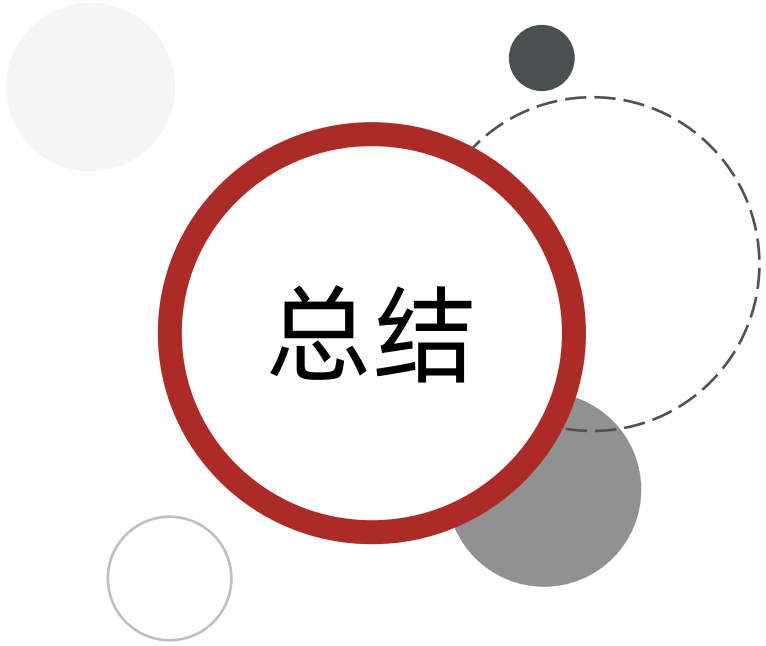
回调函数地狱

需求：展示默认第一个省，第一个城市，第一个地区在下拉菜单中

概念：在回调函数中嵌套回调函数，一直嵌套下去就形成了回调函数地狱

缺点：可读性差，异常无法捕获，耦合性严重，牵一发动全身


```
axios({ url: 'http://hmajax.itheima.net/api/province' }).then(result => {
  const pname = result.data.list[0]
  document.querySelector('.province').innerHTML = pname
  // 获取第一个省份默认下属的第一个城市名字
  axios({ url: 'http://hmajax.itheima.net/api/city', params: { pname } }).then(result => {
    const cname = result.data.list[0]
    document.querySelector('.city').innerHTML = cname
    // 获取第一个城市默认下属第一个地区名字
    axios({ url: 'http://hmajax.itheima.net/api/area', params: { pname, cname } }).then(result => {
      document.querySelector('.area').innerHTML = result.data.list[0]
    })
  })
})
```



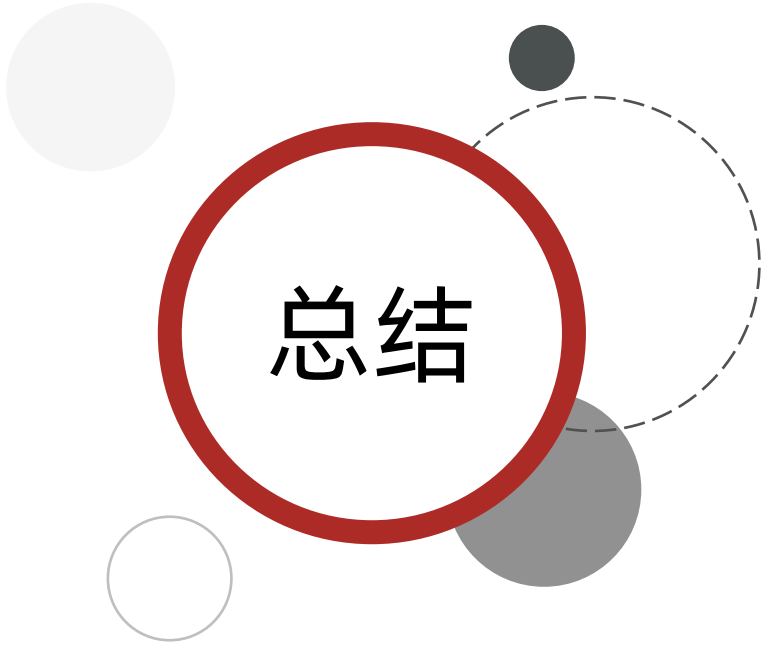
1. 什么是回调函数地狱？
   - 在回调函数一直向下嵌套回调函数，形成回调函数地狱
2. 回调函数地狱问题？
   - 可读性差
   - 异常捕获困难
   - 耦合性严重

Promise - 链式调用

概念：依靠 then() 方法会返回一个新生成的 Promise 对象特性，继续串联下一环任务，直到结束

细节：then() 回调函数中的返回值，会影响新生成的 Promise 对象最终状态和结果

好处：通过链式调用，解决回调函数嵌套问题

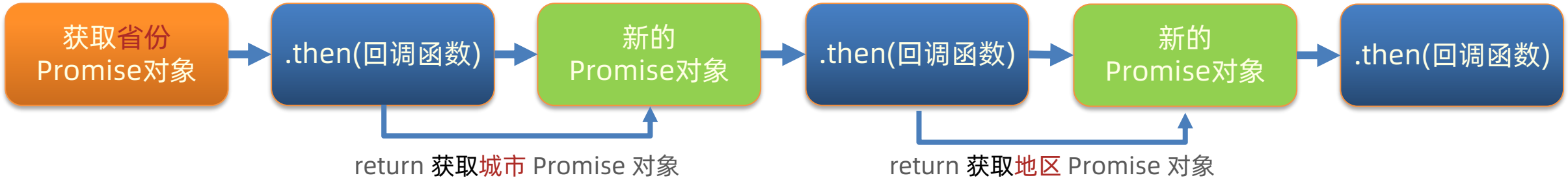


1. 什么是 Promise 的链式调用？
   - 使用 then 方法返回新 Promise 对象特性，一直串联下去
2. then 回调函数中，return 的值会传给哪里？
   - 传给 then 方法生成的新 Promise 对象
3. Promise 链式调用有什么用？
   - 解决回调函数嵌套问题

Promise 链式应用

目标：使用 Promise 链式调用，解决回调函数地狱问题

做法：每个 Promise 对象中管理一个异步任务，用 then 返回 Promise 对象，串联起来



目录

Contents

async函数和await

定义：

async 函数是使用 `async` 关键字声明的函数。async 函数是 `AsyncFunction` 构造函数的实例，并且其中允许使用 `await` 关键字。`async` 和 `await` 关键字让我们可以用一种更简洁的方式写出基于 `Promise` 的异步行为，而无需刻意地链式调用 `promise`。

概念：在 async 函数内，使用 await 关键字取代 then 函数，等待获取 Promise 对象成功状态的结果值

示例：

```
// 获取默认省市区
async function getDefaultArea() {
  const pObj = await axios({ url: 'http://hmajax.itheima.net/api/province' })
  const pname = pObj.data.list[0]
  const cObj = await axios({ url: 'http://hmajax.itheima.net/api/city', params:{ pname }})
  const cname = cObj.data.list[0]
  const aObj = await axios({ url: 'http://hmajax.itheima.net/api/area', params: { pname, cname } })
  const aname = aObj.data.list[0]
  // 赋予到页面上
  document.querySelector('.province').innerHTML = pname
  document.querySelector('.city').innerHTML = cname
  document.querySelector('.area').innerHTML = aname
}
getDefaultArea()
```

async函数和await_捕获错误

使用：

try...catch

`try...catch` 语句标记要尝试的语句块，并指定一个出现异常时抛出的响应。

语法：

```
try {
  // 要执行的代码
} catch (error) {
  // error接收的是，错误信息
  // try里代码，如果有错误，直接进入这里执行
}
```

目录

Contents

认识 - 事件循环（EventLoop）

好处：掌握 JavaScript 是如何安排和运行代码的

```
console.log(1)
setTimeout(() => {
  console.log(2)
}, 2000)
console.log(3)
```

```
console.log(1)
setTimeout(() => {
  console.log(2)
}, 0)
console.log(3)
```

事件循环（EventLoop）

概念：

> JavaScript 有一个基于**事件循环**的并发模型，事件循环负责执行代码、收集和处理事件以及执行队列中的子任务。这个模型与其它语言中的模型截然不同，比如 C 和 Java。

原因：JavaScript 单线程（某一刻只能执行一行代码），为了让耗时代码不阻塞其他代码运行，设计了事件循环模型

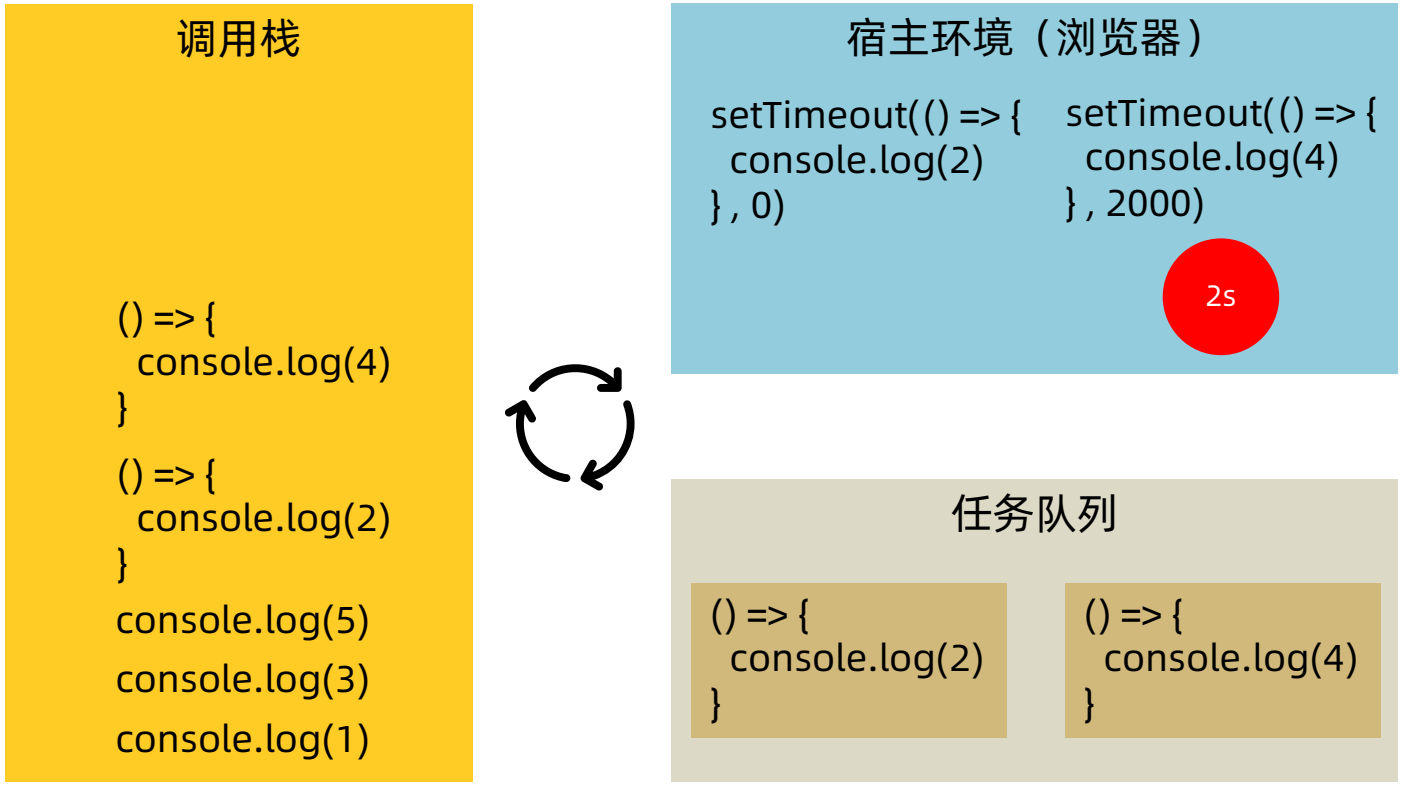
```
console.log(1)
setTimeout(() => {
  console.log(2)
}, 2000)
console.log(3)
```

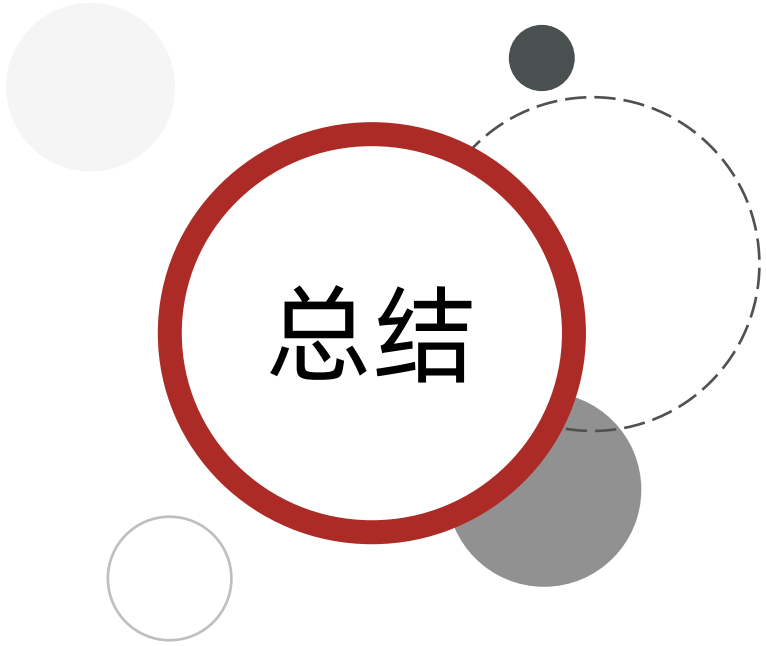
事件循环 - 执行过程

定义：执行代码和收集异步任务的模型，在调用栈空闲，反复调用任务队列里回调函数的执行机制，就叫事件循环

```
console.log(1)
setTimeout(() => {
  console.log(2)
}, 0)
console.log(3)
setTimeout(() => {
  console.log(4)
}, 2000)
console.log(5)
```

控制台输出：1　3　5　2　4



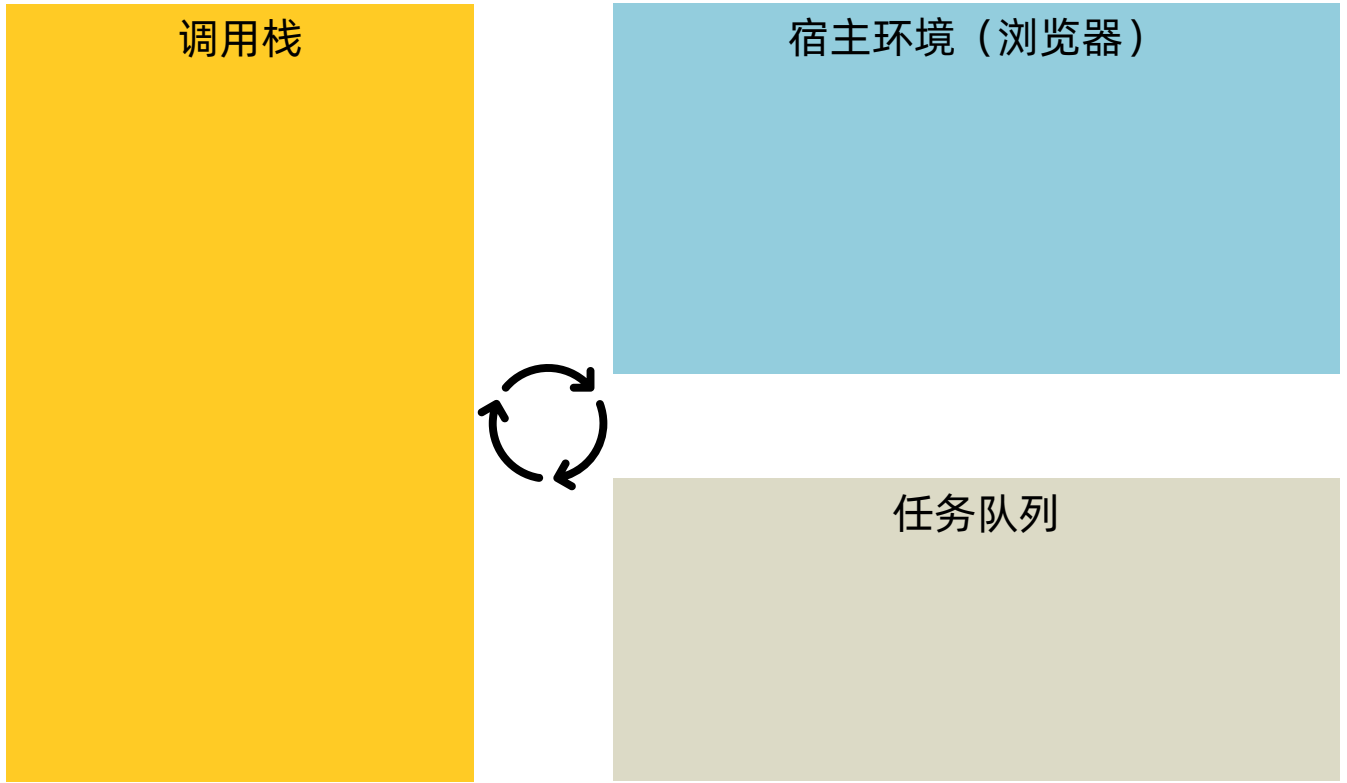


1. 什么是事件循环？
   - 执行代码和收集异步任务，在调用栈空闲时，反复调用任务队列里回调函数执行机制
2. 为什么有事件循环？
   - JavaScript 是单线程的，为了不阻塞 JS 引擎，设计执行代码的模型
3. JavaScript 内代码如何执行？
   - 执行同步代码，遇到异步代码交给宿主浏览器环境执行
   - 异步有了结果后，把回调函数放入任务队列排队
   - 当调用栈空闲后，反复调用任务队列里的回调函数

事件循环 - 练习

使用模型，分析代码执行过程

```
console.log(1)
setTimeout(() => {
  console.log(2)
}, 0)
function myFn() {
  console.log(3)
}
function ajaxFn() {
  const xhr = new XMLHttpRequest()
  xhr.open('GET', 'http://hmajax.itheima.net/api/province')
  xhr.addEventListener('loadend', () => {
    console.log(4)
  })
  xhr.send()
}
ajaxFn()
for (let i = 0; i < 1; i++) {
  console.log(5)
}
document.addEventListener('click', () => {
  console.log(6)
})
myFn()
```



宏任务与微任务

ES6 之后引入了 Promise 对象， 让 JS 引擎也可以发起异步任务

异步任务分为：

- ✓ 宏任务：由浏览器环境执行的异步代码
- ✓ 微任务：由 JS 引擎环境执行的异步代码

任务（代码）	执行所在环境
JS脚本执行事件（script）	浏览器
setTimeout/setInterval	浏览器
AJAX请求完成事件	浏览器
用户交互事件等	浏览器

任务（代码）	执行所在环境
Promise对象.then()	JS 引擎

Promise 本身是同步的，而then和catch回调函数是异步的

宏任务与微任务 - 执行顺序

使用图解-分析代码执行顺序

```
<script>
console.log(1)
setTimeout(() => {
  console.log(2)
}, 0)
const p = new Promise((resolve, reject) => {
  console.log(3)
  resolve(4)
})
p.then( result => {
  console.log(result)
})
console.log(5)
</script>
```

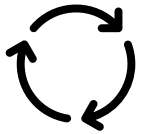
控制台打印：1　3　5　4　2

调用栈

```
() => {
  console.log(2)
}

result => {
  console.log(result)
}

console.log(5)

p.then(result => {
  console.log(result)
})

new Promise((resolve, reject)
=> {
  console.log(3)
  resolve(4)
})

console.log(1)
```

宿主环境（浏览器）

```
setTimeout(() => {
  console.log(2)
}, 0)
```



宏任务队列

```
() => {
  console.log(2)
}
```

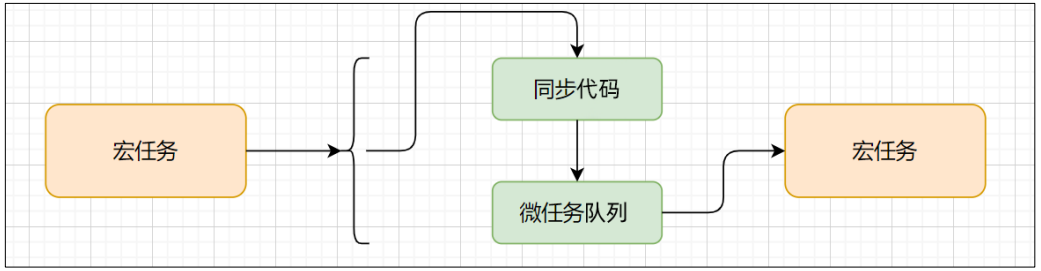
微任务队列

```
result => {
  console.log(result)
}
```

微任务队列清空后，才会执行下一个宏任务

总结

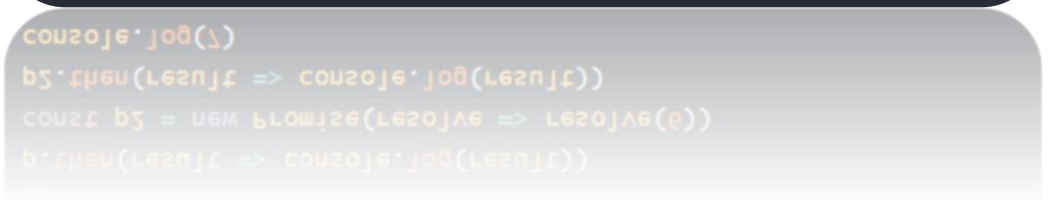
1. 什么是宏任务？
   - 浏览器执行的异步代码
   - 例如：JS 执行脚本事件，setTimeout/setInterval，AJAX请求完成事件，用户交互事件等
2. 什么是微任务？
   - JS 引擎执行的异步代码
   - 例如：Promise对象.then()的回调
3. JavaScript 内代码如何执行？
   - 执行第一个 script 脚本事件宏任务，里面同步代码
   - 遇到 宏任务/微任务 交给宿主环境，有结果回调函数进入对应队列
   - 当执行栈空闲时，清空微任务队列，再执行下一个宏任务，从1再来

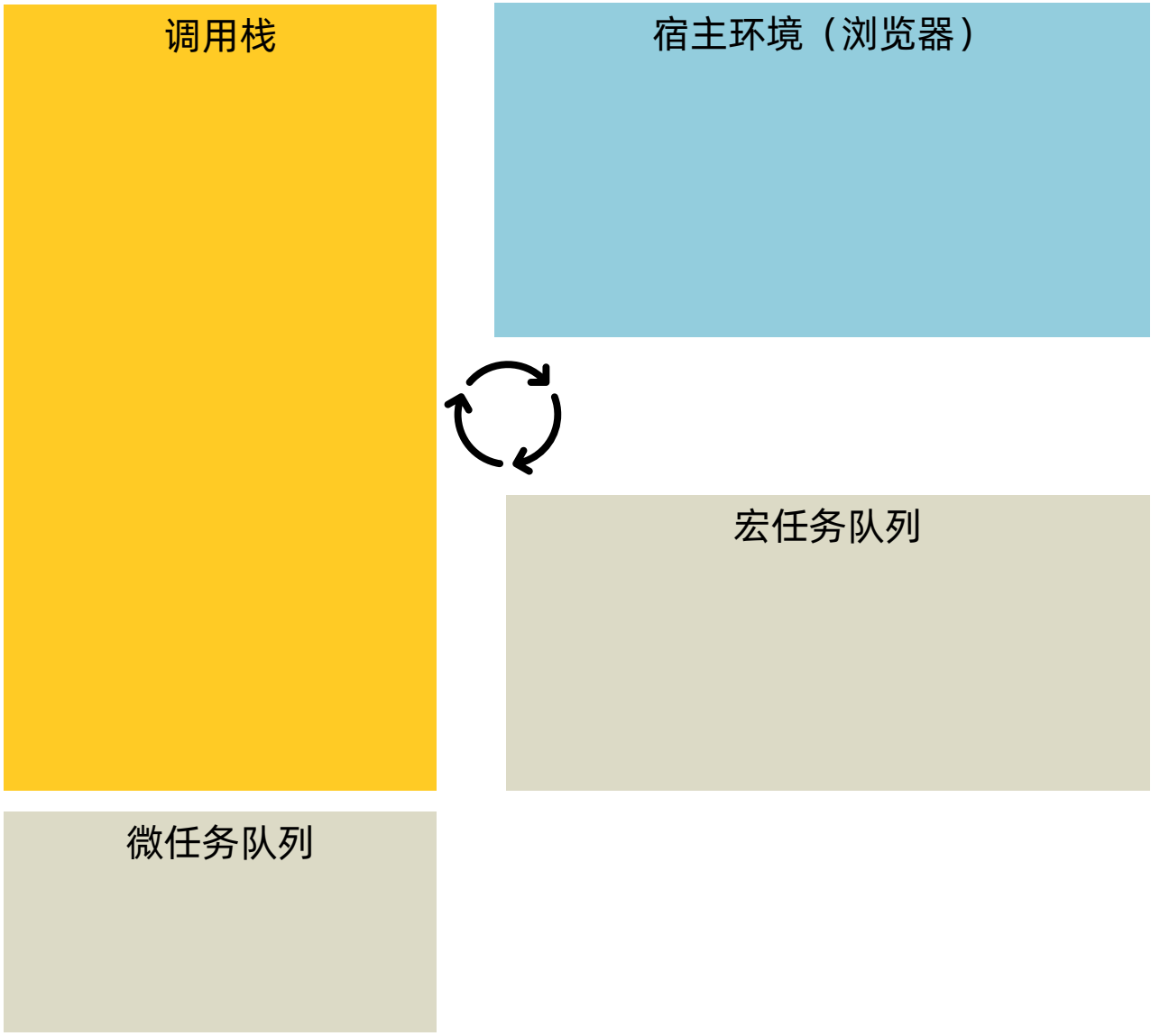


事件循环 - 经典面试题

请切换到对应配套代码，查看具体代码，并回答打印顺序

```
console.log(1)
setTimeout(() => {
  console.log(2)
  const p = new Promise(resolve => resolve(3))
  p.then(result => console.log(result))
}, 0)
const p = new Promise(resolve => {
  setTimeout(() => {
    console.log(4)
  }, 0)
  resolve(5)
})
p.then(result => console.log(result))
const p2 = new Promise(resolve => resolve(6))
p2.then(result => console.log(result))
console.log(7)
```



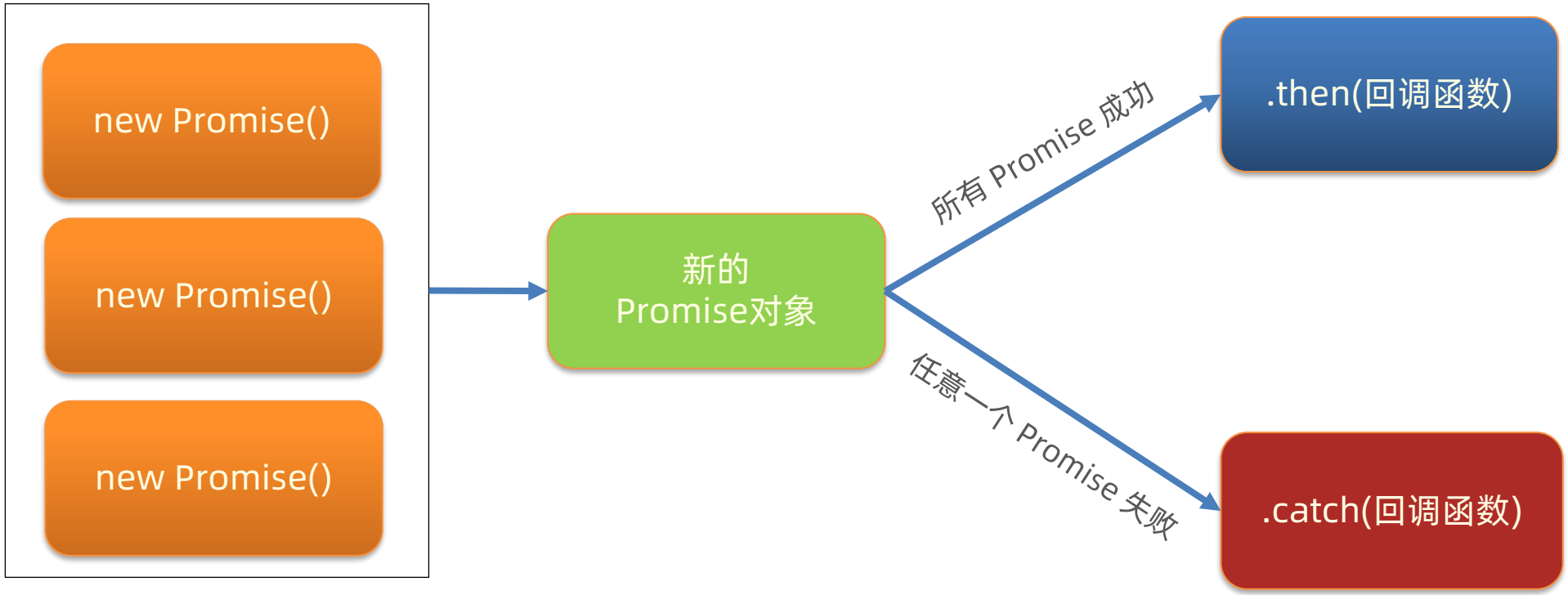


目录

Contents

Promise.all 静态方法

概念：合并多个 Promise 对象，等待所有同时成功完成（或某一个失败），做后续逻辑

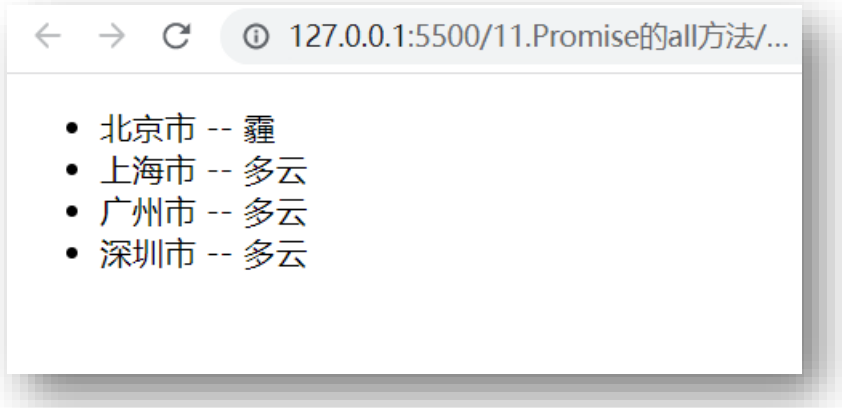


Promise.all 静态方法

语法：

```
const p = Promise.all([Promise对象，Promise对象，...])
p.then(result => {
  // result结果：[Promise对象成功结果，Promise对象成功结果，...]
}).catch(error => {
  // 第一个失败的Promise对象，抛出的异常
})
```

需求：同时请求“北京”，“上海”，“广州”，“深圳”的天气并在网页尽可能同时显示



目录

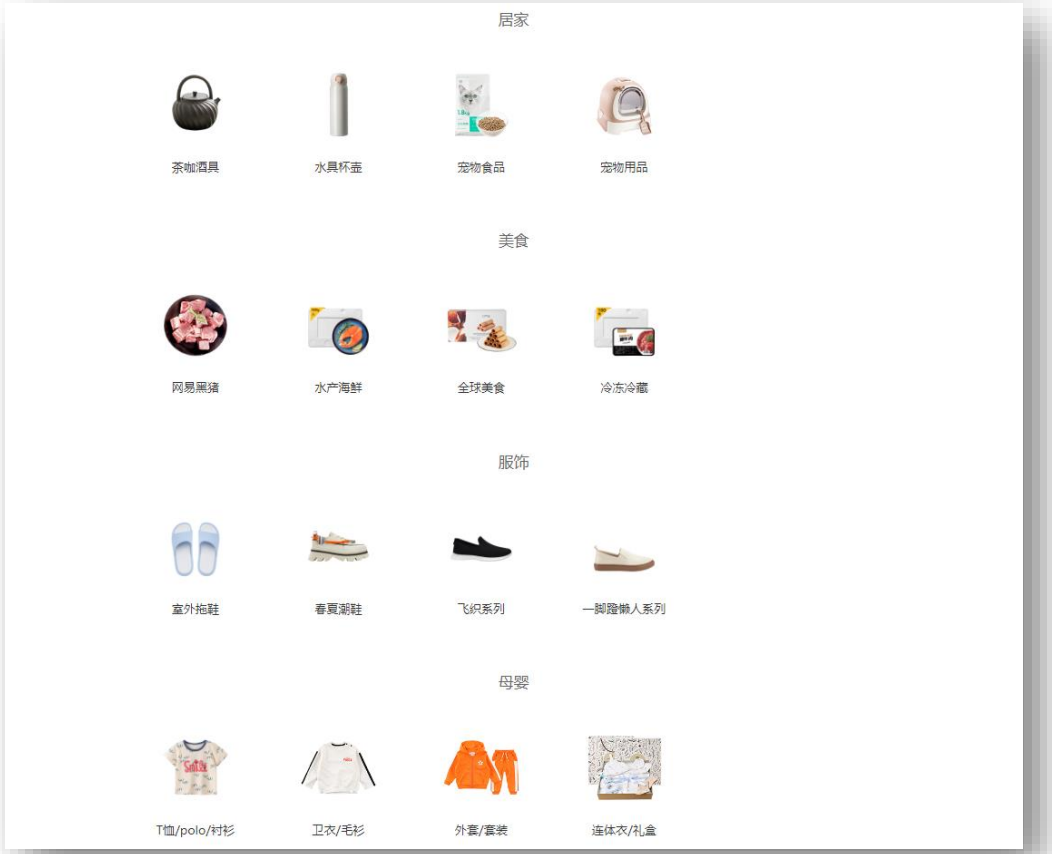
Contents

案例 商品分类

需求：尽可能同时展示所有商品分类到页面上

步骤：

1. 获取所有的一级分类数据
2. 遍历id，创建获取二级分类请求
3. 合并所有二级分类Promise对象
4. 等待同时成功，开始渲染页面



目录

Contents

案例

学习反馈 - 省市区切换

需求：完成省市区切换效果

步骤：

1. 设置省份数据到下拉菜单
2. 切换省份，设置城市数据到下拉菜单，并清空地区下拉菜单
3. 切换城市，设置地区数据到下拉菜单

案例

学习反馈 - 数据提交

需求：收集学习反馈数据，提交保存

步骤：

1. 监听提交按钮的点击事件
2. 依靠插件收集表单数据
3. 基于 axios 提交保存，显示结果

传智教育旗下高端IT教育品牌