

AJAX



黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌



同步代码和异步代码

同步和异步代码

同步代码: 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，在将来完成后触发一个**回调函数**

同步和异步代码

```
console.log(1)
const num = 1 + 1
console.log(num)
```

直接输出: 1,2

```
console.log(1)
setTimeout(() => {
  console.log(2)
}, 1000)
console.log(3)
```

直接输出: 1,3
一秒之后: 2

```
console.log(1)
document.body.addEventListener('click', () => {
  console.log(2)
})
console.log(3)
```

直接输出: 1,3
点击之后: 2

```
console.log(1)
const xhr = new XMLHttpRequest()
xhr.open('get', 'url地址')
xhr.addEventListener('loadend', () => {
  console.log(2)
})
xhr.send()
console.log(3)
```

直接输出: 1,3
响应回来: 2



总结

1. 同步代码:

逐行执行, 需原地等待结果后, 才继续往下执行

2. 异步代码:

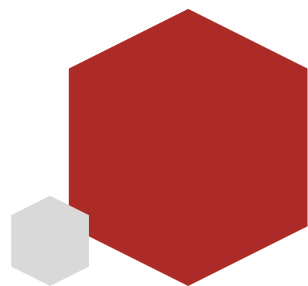
调用后耗时, 不阻塞代码继续执行, 在将来完成后触发回调函数

3. 常见异步代码:

定时器, 事件, XHR

4. 如何接收异步代码结果:

回调函数



回调函数地狱

回调函数地狱

回调函数地狱: 在回调函数中调用回调函数，形成的**代码结构**称之为回调函数地狱

缺点: 可读性差, 异常捕获困难

```
axios({ url: '地址1' }).then(result => {  
  // 第一次请求完成  
  // 其他代码略  
  axios({ url: '地址2' }).then(result => {  
    // 第二次请求完成  
    // 其他代码略  
    axios({ url: '地址3' }).then(result => {  
      // 第三次请求完成  
      // 其他代码略。。。  
    })  
  })  
})  
})
```

试一试

需求: 展示数据到下拉框中

1. 获取省份数据并展示第1个省
2. 获取第1个省的城市数据,并展示第1个城市
3. 获取第1个城市的区数据, 并展示第1个区

回调函数地狱

省份: 城市: 地区:



总结

1. 回调函数地狱:

在回调函数中调用回调函数，形成的代码结构

2. 回调函数地狱缺点:

可读性差，异常捕获困难



Promise-链式调用

Promise-链式调用

Promise链式调用: 每一个then方法还会返回一个新生成的promise对象，这个对象可被用作链式调用

then方法的返回值: then方法中的回调函数的返回值，会影响新生成的Promise对象最终状态和结果



```
promise对象
.then(() => {
  // 略
}).then(()=>{
  // 略
}).then(()=>{
  // 略
})
```



总结

1. Promise的链式调用:

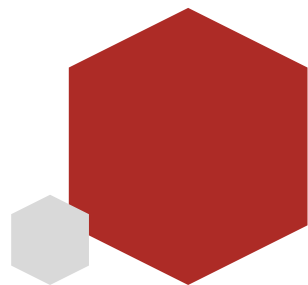
then方法会返回一个新的Promise对象，这个对象可以被链式调用

2. then的回调函数中，return的值会传递给谁?

then方法返回的新的Promise对象

3. Promise链式调用的作用?

解决回调函数嵌套（回调函数地狱）



Promise-链式调用-解决回调地狱

Promise-链式调用-解决回调地狱

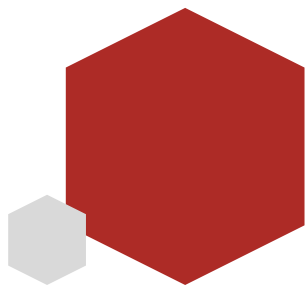
需求:展示数据到下拉框中（使用Promise-链式编程完成）

核心步骤: then的回调函数中返回Promise对象

Promise-链式调用-解决回调地狱

省份: 城市: 地区:





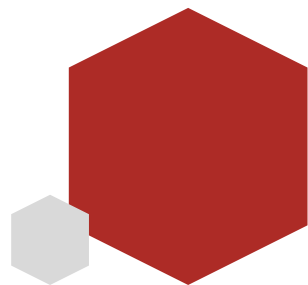
async函数和await

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。

语法: 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

```
async function func() {  
  const res1 = await Promise对象1  
  const res2 = await Promise对象2  
  const res3 = await Promise对象3  
}  
func()
```

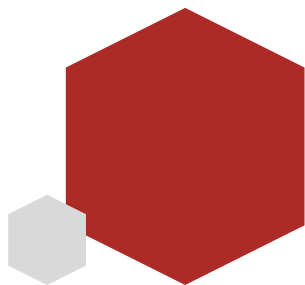



async函数和await-错误捕获

async函数和await-错误捕获

定义: 使用 `async/await` 关键字就可以在异步代码中使用普通的 `try/catch` 代码块。

```
try {  
    // 需要被执行的语句  
} catch (error) {  
    // error 接收错误信息  
    // try 有错误时执行的语句  
}
```



事件循环-eventloop

事件循环-eventloop

定义: 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。

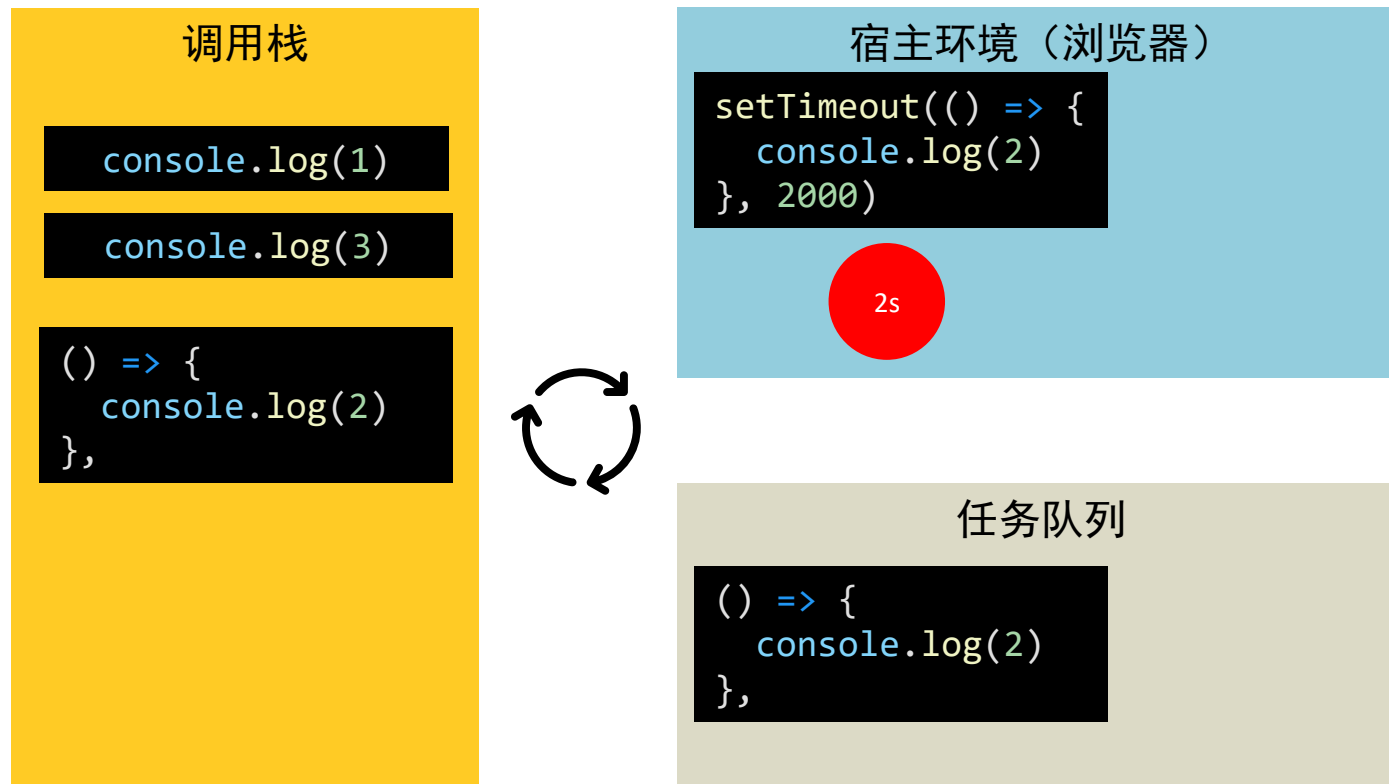
JavaScript是单线程（代码逐行执行）的，为了不让耗时代码阻塞其他代码运行，就设计了事件循环

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

事件循环-执行过程

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

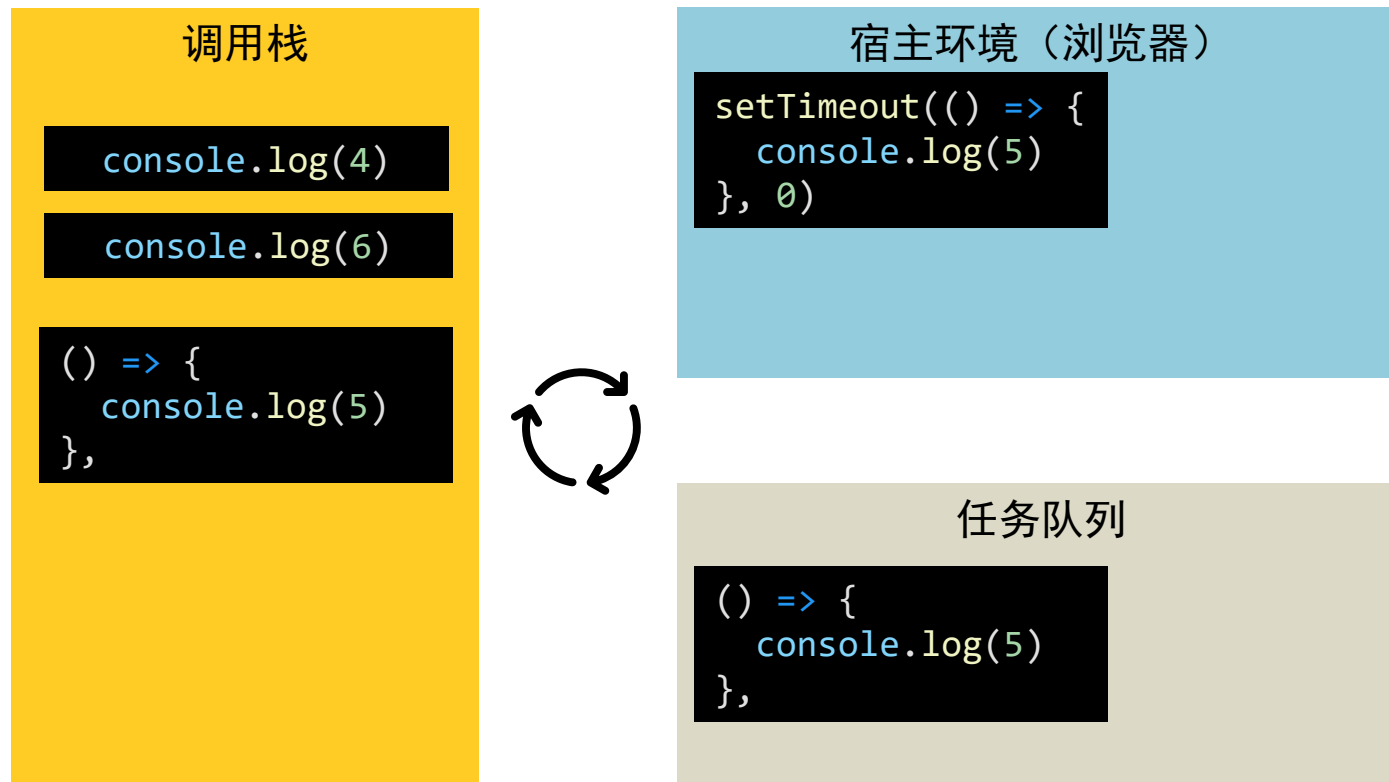
输出: 1 3 2



事件循环-执行过程

```
console.log(4)
setTimeout(() => {
  console.log(5)
}, 0)
console.log(6)
```

输出: 4 6 5



总结

1. 事件循环:

JavaScript代码的**执行机制**

2. 为什么有事件循环:

JavaScript是**单线程**的，为了**不让耗时代码阻塞**其他代码执行，设计的执行代码的模型（机制）

3. 事件循环的模型（执行过程）:

调用栈执行同步代码，**异步代码**交给宿主环境执行

异步代码等待时机成熟，送入**任务队列排队**

调用栈**空闲**时，反复**查看并调用**任务队列里的回调函数



事件循环-练习

事件循环-练习

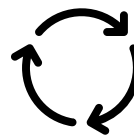
需求: 分析代码执行过程

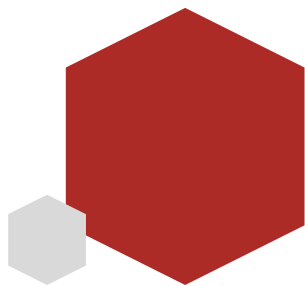
```
console.log(1)
setTimeout(() => {
  console.log(2)
}, 0)
setTimeout(() => {
  console.log(3)
}, 0)
function getProvince() {
  const xhr = new XMLHttpRequest()
  xhr.open('get', 'http://hmajax.itheima.net/api/province')
  xhr.addEventListener('loadend', () => {
    console.log(4)
  })
  xhr.send()
}
getProvince()
document.addEventListener('click', () => {
  console.log(5)
})
console.log(6)
```

调用栈

宿主环境（浏览器）

任务队列





(宏) 任务和微任务

(宏) 任务与微任务

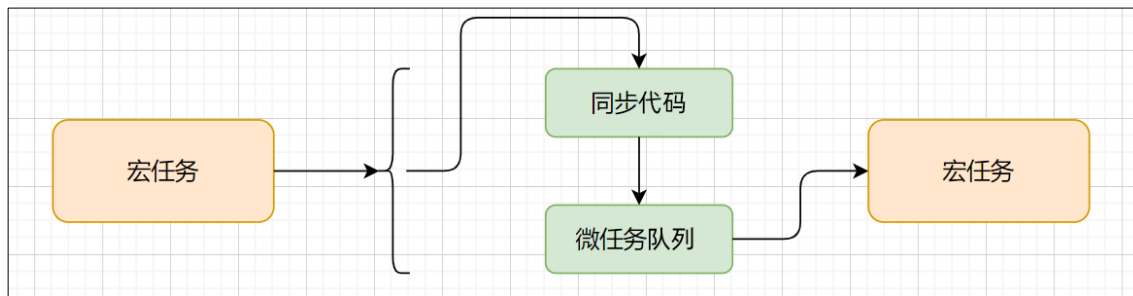
定义:每个代理都是由事件循环驱动的，事件循环负责收集用事件（包括用户事件以及其他非用户事件等）、对任务进行排队以便在合适的时候执行回调。然后它执行所有处于等待中的 JavaScript 任务（宏任务），然后是微任务，然后在开始下一次循环之前执行一些必要的渲染和绘制操作。

异步任务:

1. (宏) 任务: 由浏览器环境执行的异步代码
2. 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()和catch()	JS 引擎



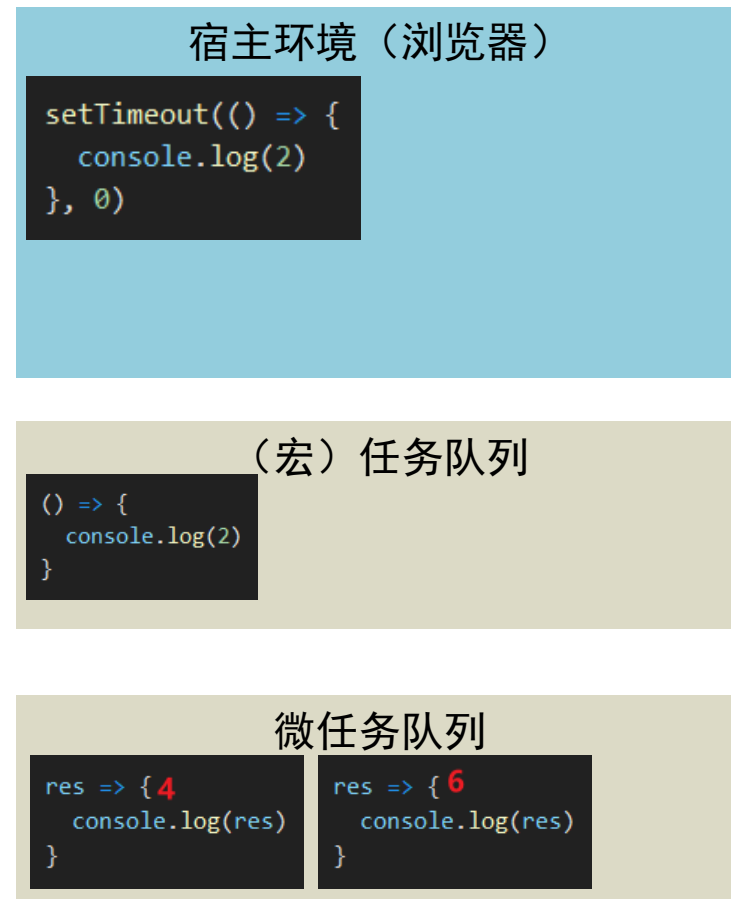
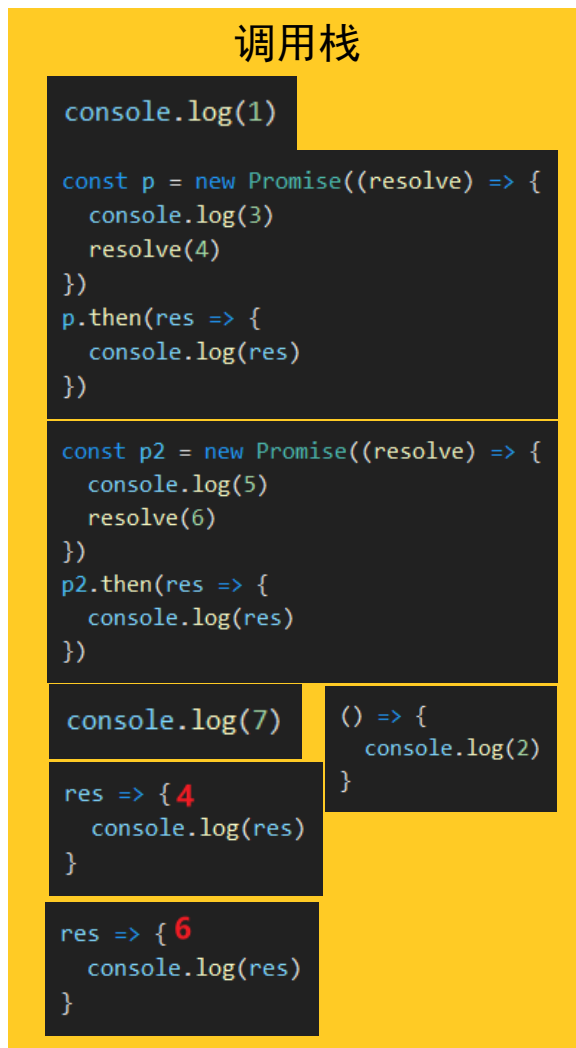
1. 执行第一个script脚本事件的宏任务中的同步代码
2. (宏)任务和微任务的回调函数，进入对应队列
3. 调用栈空闲时，清空微任务队列，再执行下一个宏任务

注: Promise本身是同步的，then和catch回调函数是异步的

(宏) 任务与微任务

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

输出: 1 3 5 7 4 6 2





总结

1. (宏) 任务:

浏览器执行的异步代码: script标签, 定时器, AJAX请求完成事件, 用户交互事件等

2. 微任务:

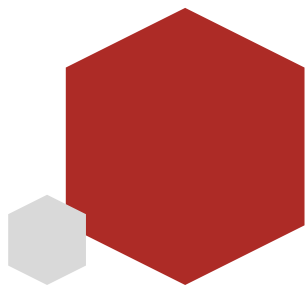
JS引擎执行的异步代码: Promise对象.then和catch的回调

3. 事件循环的模型 (执行过程):

执行第一个script脚本事件的宏任务中的同步代码

(宏) 任务和微任务的回调函数, 进入对应队列

调用栈空闲时, 清空微任务队列, 再执行下一个宏任务



事件循环-经典面试题

事件循环-经典面试题

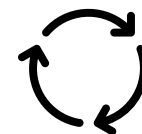
```
<script>  
  console.log(1)  
  setTimeout(() => {  
    console.log(2)  
    const p = new Promise((resolve) => resolve(3))  
    p.then(res => console.log(res))  
  }, 0)  
  const p = new Promise(resolve => {  
    setTimeout(() => console.log(4), 0)  
    resolve(5)  
  })  
  p.then(res => {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

调用栈

宿主环境（浏览器）

（宏）任务队列

微任务队列



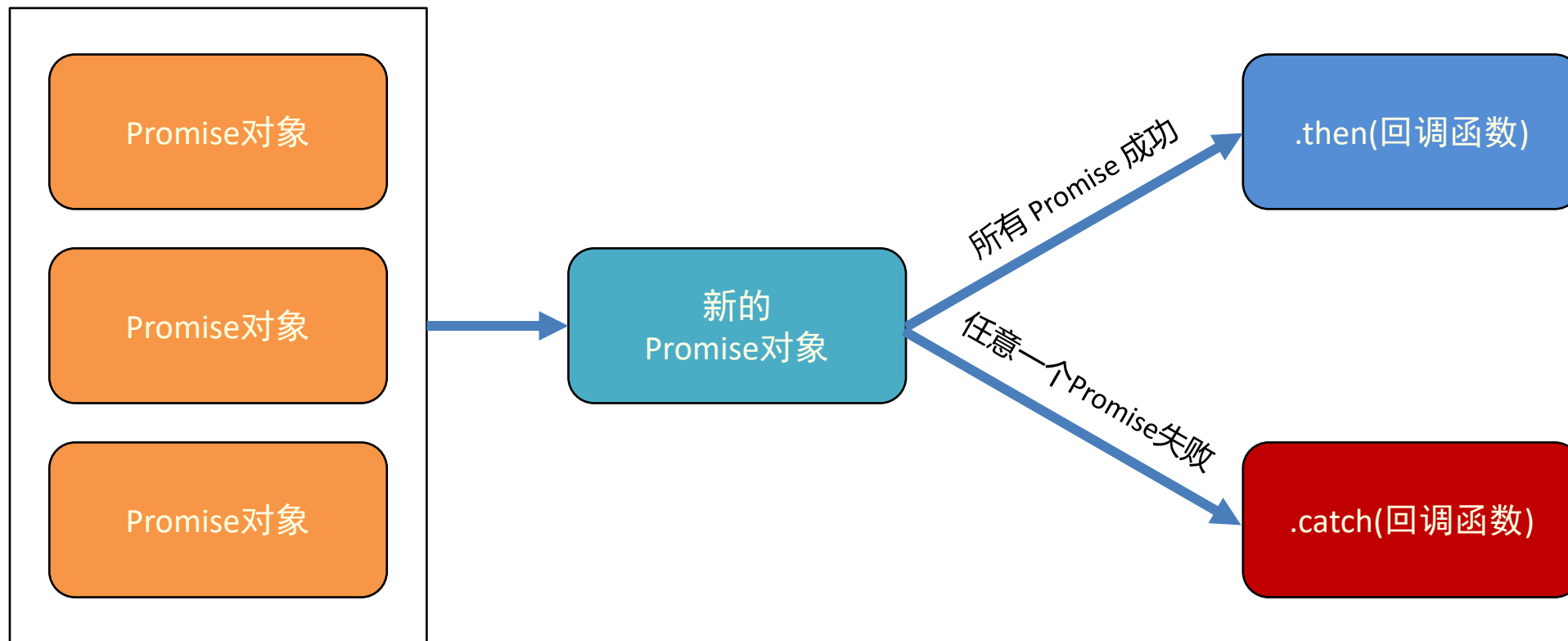


Promise.all 静态方法

Promise.all 静态方法

说明: 此方法在集合多个promise的返回结果时很有用

作用: 将多个Promise对象包装成一个新的Promise对象，获取**所有的**成功结果，或**某一个的**失败原因



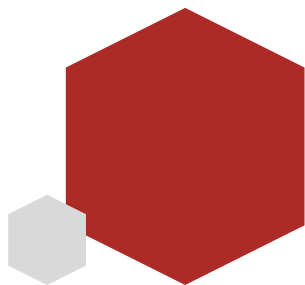
Promise.all 静态方法

语法:

```
const p = Promise.all([Promise对象1, Promise对象2...])
p.then(results => {
  // results: 结果数组[Promise对象1的结果, Promise对象2的结果]
}).catch(err => {
  // err: 第一个失败的Promise抛出的异常
})
```

需求: 查询“北京”、“上海”、“广州”、“深圳”的天气, 并在获取到**所有结果**之后, 渲染到页面上

- 北京市 -- 霾
- 上海市 -- 多云
- 广州市 -- 多云
- 深圳市 -- 多云



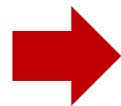
案例-商品分类

需求

获取**所有分类**数据并**同时**渲染到页面上



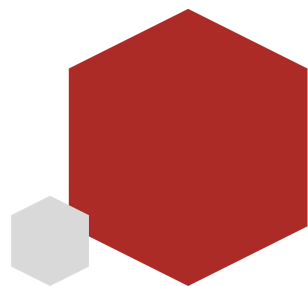
获取一级商品分类



获取所有二级商品分类



渲染到页面上



案例-学习反馈

需求

1. 省份列表
2. 城市列表
3. 地区列表
4. 反馈提交

学习反馈

 黑马程序员
www.itheima.com



关注黑马，掌握更多前端资讯



热门校区 北京 上海 广州 深圳


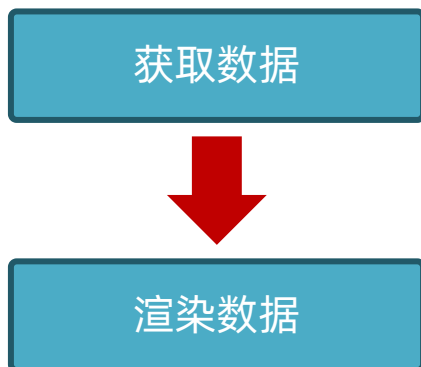
地区选择

您的称呼

宝贵建议

省份列表

获取省份信息，并渲染



地区选择	省份	城市	地区
您的称呼	省份		
	北京		
	天津		
	河北省		

需求

1. 省份列表
2. 城市列表
3. 地区列表
4. 反馈提交

学习反馈

 **黑马程序员**
www.itheima.com




关注黑马，掌握更多前端资讯

热门校区 北京 上海 广州 深圳

地区选择

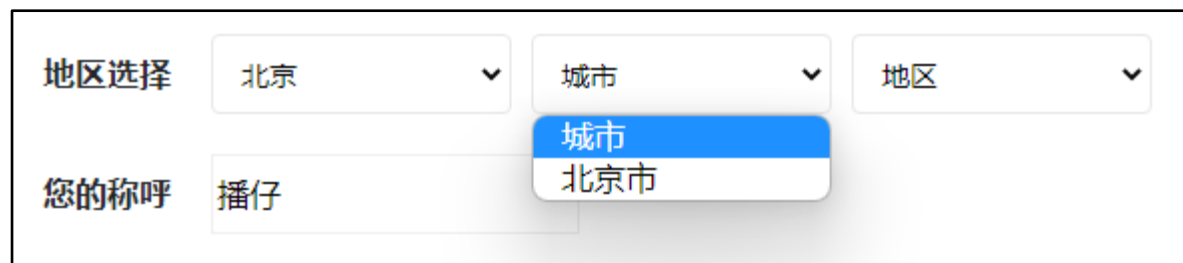
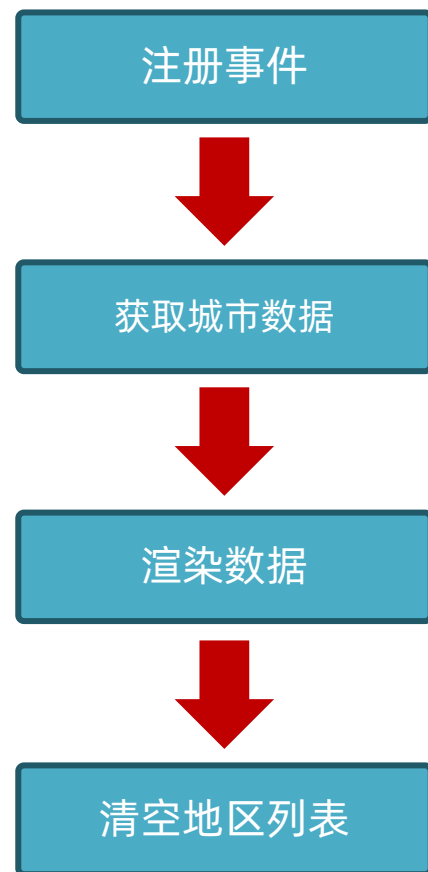
您的称呼

宝贵建议



城市列表

选择省份之后，渲染城市列表，清空地区列表



地区选择 北京 城市 地区

您的称呼 播仔

城市
北京市

需求

1. 省份列表
2. 城市列表
3. 地区列表
4. 反馈提交

学习反馈

 黑马程序员
www.itheima.com



关注黑马，掌握更多前端资讯

热门校区 北京 上海 广州 深圳

地区选择

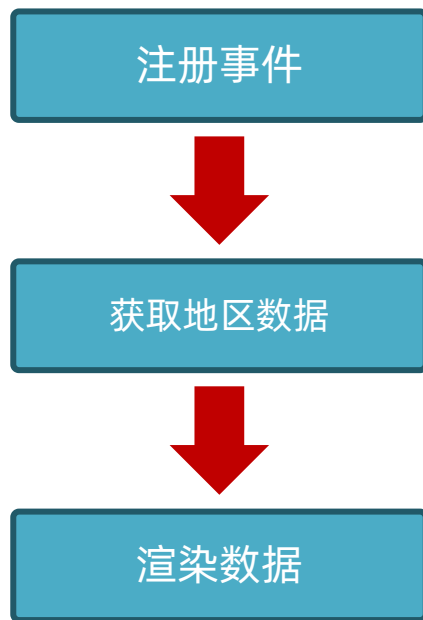
您的称呼

宝贵建议



地区列表

选择城市之后，渲染地区列表



地区选择: 北京 (dropdown), 北京市 (dropdown), 地区 (dropdown)

您的称呼: 播仔 (input)

您对AJAX阶段课程宝贵的建议 (input)

地区列表 (dropdown menu):

- 地区
- 东城区
- 西城区
- 朝阳区
- 丰台区

需求

1. 省份列表
2. 城市列表
3. 地区列表
4. 反馈提交

学习反馈

 黑马程序员
www.itheima.com




关注黑马，掌握更多前端资讯

热门校区 北京 上海 广州 深圳

地区选择

您的称呼

宝贵建议



反馈提交

点击提交按钮，提交反馈信息到服务器



热门校区 北京 上海 广州 深圳

地区选择 省份 城市 地区

您的称呼

宝贵建议



传智教育旗下高端IT教育品牌