

看了就会，手写Promise原理，最通俗易懂的版本！！！！



Sunshine_Lin Lv6

2021年08月10日 00:39 · 阅读 32714

+ 关注

 897

 98

 收藏



@稀土掘金技术社区

前言

👍 897

💬 98

★ 收藏

大家好，我是林三心，相信大家在日常开发中都用过[Promise](#)，我一直有个梦想，就是[以最通俗的话，讲最复杂的知识](#)，所以我把[通俗易懂](#)放在了首位，今天就带大家手写实现以下[Promise吧](#)，相信大家一看就懂。



resolve和reject

咱们来看一段Promise的代码：

```
let p1 = new Promise((resolve, reject) => {  
  resolve('成功')  
  reject('失败')  
})  
console.log('p1', p1)
```

js 复制代码



897



98



收藏

```
    resolve('成功')
  })
  console.log('p2', p2)

  let p3 = new Promise((resolve, reject) => {
    throw('报错')
  })
  console.log('p3', p3)
```

那么会输出什么呢？请看：



897



98



收藏

```
p1 ▼ Promise {<fulfilled>: "成功"} ⓘ  
  ► [[Prototype]]: Promise  
    [[PromiseState]]: "fulfilled"  
    [[PromiseResult]]: "成功"
```

```
p2 ▼ Promise {<rejected>: "失败"} ⓘ  
  ► [[Prototype]]: Promise  
    [[PromiseState]]: "rejected"  
    [[PromiseResult]]: "失败"
```

```
p3 ▼ Promise {<rejected>: "报错"} ⓘ  
  ► [[Prototype]]: Promise  
    [[PromiseState]]: "rejected"  
    [[PromiseResult]]: "报错"@稀土掘金技术社区
```

这里暴露出了四个知识点：

- 1、执行了 `resolve` ，Promise状态会变成 `fulfilled`
- 2、执行了 `reject` ，Promise状态会变成 `rejected`
- 3、Promise只以 `第一次为准` ，第一次成功就 `永久` 为 `fulfilled` ，第一次失败就永远状态为 `rejected`
- 4、Promise中有 `throw` 的话，就相当于执行了 `reject`



897



98



收藏

1、实现resolve与reject

大家要注意：Promise的初始状态是 `pending`

这里很重要的一步是 `resolve和reject的绑定this`，为什么要绑定 `this` 呢？这是为了resolve和reject的 `this指向` 永远指向当前的 `MyPromise实例`，防止随着函数执行环境的改变而改变

```
class MyPromise {  
  // 构造方法  
  constructor(executor) {  
  
    // 初始化值  
    this.initValue()  
    // 初始化this指向  
    this.initBind()  
    // 执行传进来的函数  
    executor(this.resolve, this.reject)  
  }  
  
  initBind() {  
    // 初始化this  
    this.resolve = this.resolve.bind(this)  
    this.reject = this.reject.bind(this)  
  }  
  
  initValue() {
```

js 复制代码



```
    this.PromiseState = 'pending' // 状态
  }

  resolve(value) {
    // 如果执行resolve, 状态变为fulfilled
    this.PromiseState = 'fulfilled'
    // 终值为传进来的值
    this.PromiseResult = value
  }

  reject(reason) {
    // 如果执行reject, 状态变为rejected
    this.PromiseState = 'rejected'
    // 终值为传进来的reason
    this.PromiseResult = reason
  }
}
```

咱们来测试一下代码吧:

```
const test1 = new MyPromise((resolve, reject) => {
  resolve('成功')
})
console.log(test1) // MyPromise { PromiseState: 'fulfilled', PromiseResult: '成功' }

const test2 = new MyPromise((resolve, reject) => {
  reject('失败')
})
console.log(test2) // MyPromise { PromiseState: 'rejected', PromiseResult: '失败' }
```

js 复制代码



2. 状态不可变

其实上面的代码是有问题的，什么问题呢？看看：

```
const test1 = new MyPromise((resolve, reject) => {  
  resolve('成功')  
  reject('失败')  
})  
  
console.log(test1) // MyPromise { PromiseState: 'rejected', PromiseResult: '失败' }
```

js 复制代码

正确的应该是状态为 `fulfilled`，结果是 `成功`，这里明显没有 `以第一次为准`

之前说了，Promise只以 `第一次为准`，第一次成功就 `永久` 为 `fulfilled`，第一次失败就永远状态为 `rejected`，具体是什么流程呢？我给大家画了一张图：

Promise有三种状态：

- `pending`：等待中，是初始状态
- `fulfilled`：成功状态
- `rejected`：失败状态



897

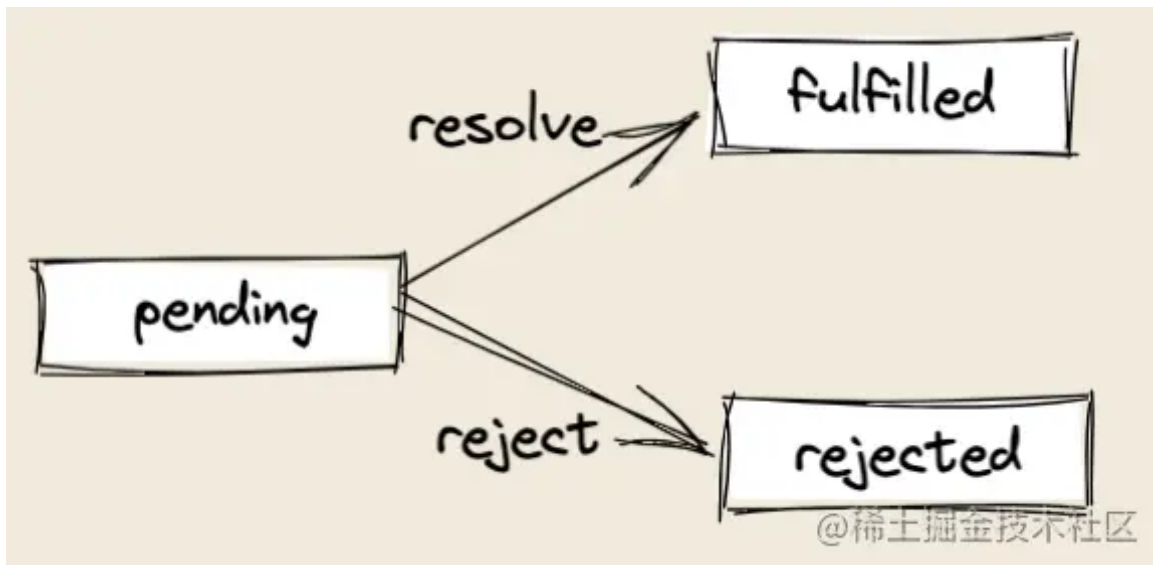


98



收藏

一旦状态从 `pending` 变为 `fulfilled`或者`rejected`，那么此Promise实例的状态就定死了。



其实实现起来也很容易，加个判断条件就行：

```
resolve(value) {  
  // state是不可变的  
  + if (this.PromiseState !== 'pending') return  
  // 如果执行resolve，状态变为fulfilled  
  this.PromiseState = 'fulfilled'  
  // 终值为传进来的值  
  this.PromiseResult = value  
}  
  
reject(reason) {  
  // state是不可变的
```

js 复制代码

👍 897

💬 98

★ 收藏

```
this.PromiseState = 'rejected'  
// 终值为传进来的reason  
this.PromiseResult = reason  
}
```

再来看看效果：

```
const test1 = new MyPromise((resolve, reject) => {  
  // 只以第一次为准  
  resolve('成功')  
  reject('失败')  
})  
console.log(test1) // MyPromise { PromiseState: 'fulfilled', PromiseResult: '成功' }
```

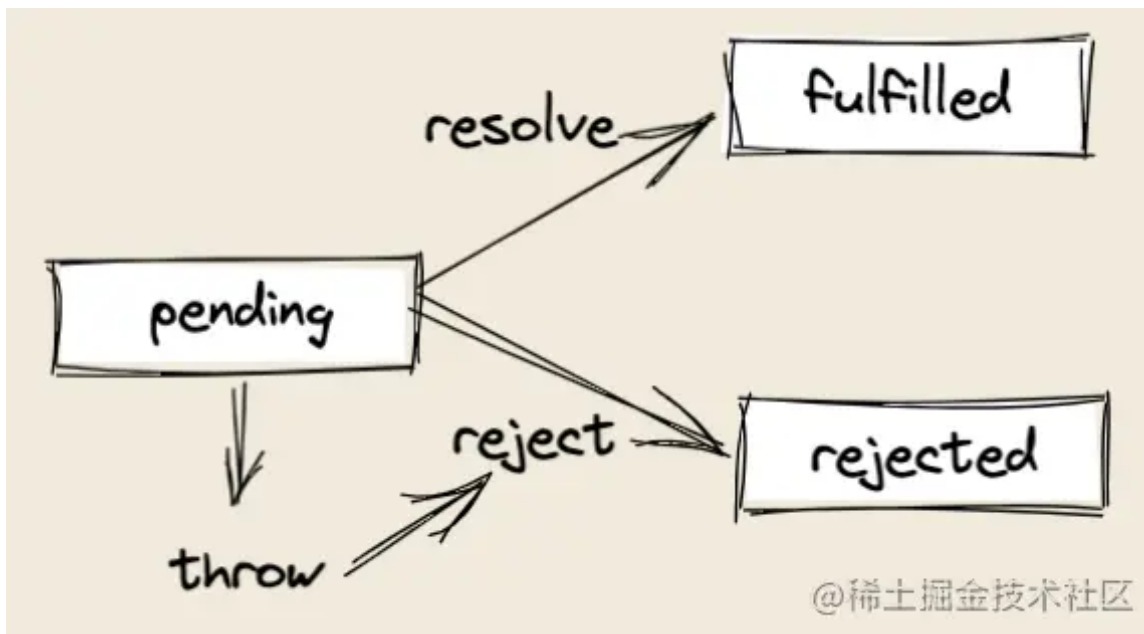
js 复制代码

3. throw

 897

 98

 收藏



Promise中有 `throw` 的话，就相当于执行了 `reject`。这就要使用 `try catch` 了

```
+      try {  
+        // 执行传进来的函数  
+        executor(this.resolve, this.reject)  
+      } catch (e) {  
+        // 捕捉到错误直接执行reject  
+        this.reject(e)  
+      }
```

js 复制代码

咱们来看看效果：

js 复制代码

👍 897

💬 98

★ 收藏

```
})  
console.log(test3) // MyPromise { PromiseState: 'rejected', PromiseResult: '失败' }
```

then

咱们平时使用then方法是这么用的：

```
// 马上输出 "成功"  
const p1 = new Promise((resolve, reject) => {  
  resolve('成功')  
}).then(res => console.log(res), err => console.log(err))  
  
// 1秒后输出 "失败"  
const p2 = new Promise((resolve, reject) => {  
  setTimeout(() => {  
    reject('失败')  
  }, 1000)  
}).then(res => console.log(res), err => console.log(err))  
  
// 链式调用 输出 200  
const p3 = new Promise((resolve, reject) => {  
  resolve(100)  
}).then(res => 2 * res, err => console.log(err))  
  .then(res => console.log(res), err => console.log(err))
```

js 复制代码

可以总结出这几个知识点：



897



98



收藏

- 当Promise状态为 `fulfilled` 执行 成功回调 ，为 `rejected` 执行 失败回调
- 如resolve或reject在定时器里， 则定时器结束后再执行then
- then支持 链式调用 ，下一次then执行 受上一次then返回值的影响

下面咱们就一步一步地去实现他吧

1. 实现then

js 复制代码

```
then(onFulfilled, onRejected) {  
  // 接收两个回调 onFulfilled, onRejected  
  
  // 参数校验, 确保一定是函数  
  onFulfilled = typeof onFulfilled === 'function' ? onFulfilled : val => val  
  onRejected = typeof onRejected === 'function' ? onRejected : reason => { throw reason }  
  
  if (this.PromiseState === 'fulfilled') {  
    // 如果当前为成功状态, 执行第一个回调  
    onFulfilled(this.PromiseResult)  
  } else if (this.PromiseState === 'rejected') {  
    // 如果当前为失败状态, 执行第二个回调  
    onRejected(this.PromiseResult)  
  }  
  
}
```

 897

 98

 收藏

```
// 输出 "成功"

const test = new MyPromise((resolve, reject) => {
  resolve('成功')
}).then(res => console.log(res), err => console.log(err))
```

2. 定时器情况

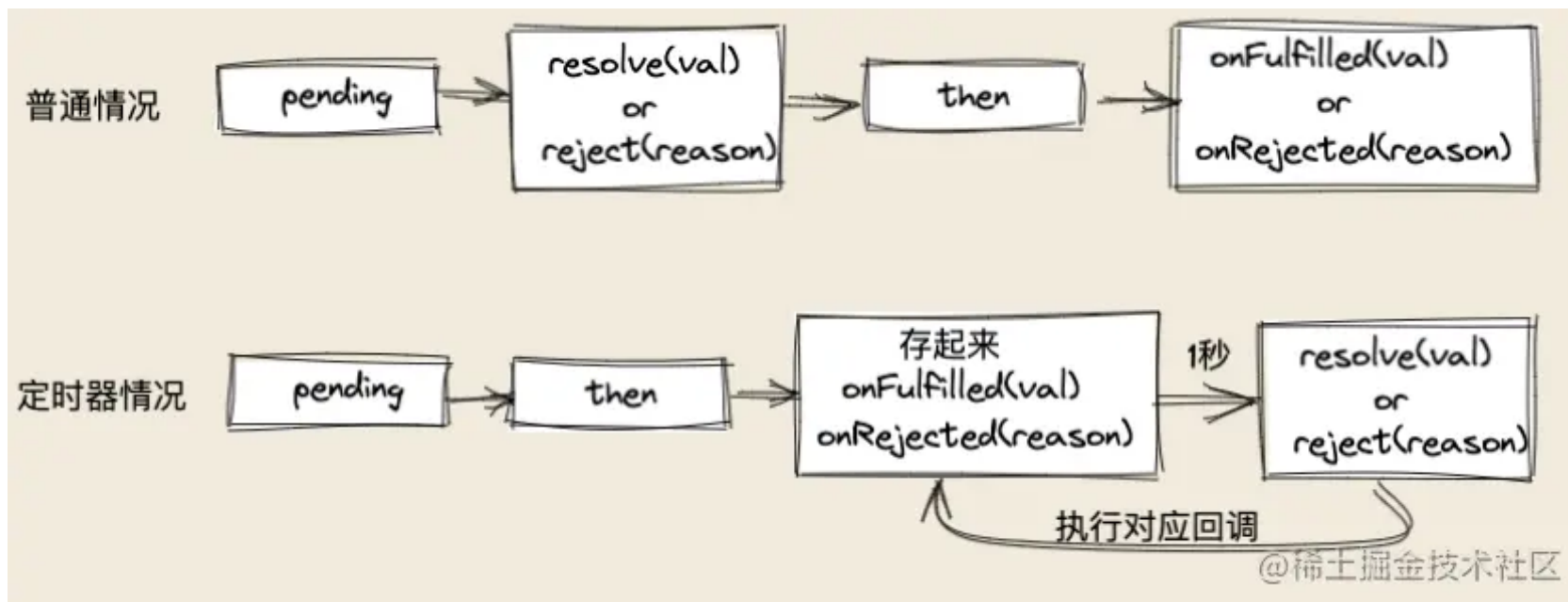
上面我们已经实现了 `then` 的基本功能。那如果是 **定时器** 情况呢？

还是那个代码，怎样才能保证，1秒后才执行then里的失败回调呢？

```
// 1秒后输出 "成功"

const p2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject('失败')
  }, 1000)
}).then(res => console.log(res), err => console.log(err))
```

我们不能确保1秒后才执行then函数，但是我们可以保证1秒后再执行then里的回调，可能这里大家有点懵逼，我同样用一张图给大家讲讲吧：



也就是在这1秒时间内，我们可以先把then里的两个回调保存起来，然后等到1秒过后，执行了resolve或者reject，咱们再去判断状态，并且判断要去执行刚刚保存的两个回调中的哪一个回调。

那么问题来了，我们怎么知道当前1秒还没走完甚至还没开始走呢？其实很好判断，只要状态是 `pending`，那就证明定时器还没跑完，因为如果定时器跑完的话，那状态肯定就不是 `pending`，而是 `fulfilled或者rejected`

那是用什么来保存这些回调呢？建议使用 `数组`，因为一个promise实例可能会 `多次then`，用数组就一个一个保存了

```
initValue() {  
  // 初始化值  
  this.PromiseResult = null // 终值  
  this.PromiseState = 'pending' // 状态
```

js 复制代码

👍 897

💬 98

★ 收藏

```

}

resolve(value) {
  // state是不可变的
  if (this.PromiseState !== 'pending') return
  // 如果执行resolve, 状态变为fulfilled
  this.PromiseState = 'fulfilled'
  // 终值为传进来的值
  this.PromiseResult = value
  // 执行保存的成功回调
+   while (this.onFulfilledCallbacks.length) {
+     this.onFulfilledCallbacks.shift()(this.PromiseResult)
+   }
}

reject(reason) {
  // state是不可变的
  if (this.PromiseState !== 'pending') return
  // 如果执行reject, 状态变为rejected
  this.PromiseState = 'rejected'
  // 终值为传进来的reason
  this.PromiseResult = reason
  // 执行保存的失败回调
+   while (this.onRejectedCallbacks.length) {
+     this.onRejectedCallbacks.shift()(this.PromiseResult)
+   }
}

then(onFulfilled, onRejected) {
  // 接收两个回调 onFulfilled, onRejected

```



897



98



收藏


```

onFulfilled = typeof onFulfilled === 'function' ? onFulfilled : val => val
onRejected = typeof onRejected === 'function' ? onRejected : reason => { throw reason }

if (this.PromiseState === 'fulfilled') {
    // 如果当前为成功状态, 执行第一个回调
    onFulfilled(this.PromiseResult)
} else if (this.PromiseState === 'rejected') {
    // 如果当前为失败状态, 执行第二哥回调
    onRejected(this.PromiseResult)
+   } else if (this.PromiseState === 'pending') {
+       // 如果状态为待定状态, 暂时保存两个回调
+       this.onFulfilledCallbacks.push(onFulfilled.bind(this))
+       this.onRejectedCallbacks.push(onRejected.bind(this))
+   }
}

```

加完上面的代码, 咱们来看看定时器的效果吧:

```

const test2 = new MyPromise((resolve, reject) => {
    setTimeout(( ) => {
        resolve('成功') // 1秒后输出 成功
        // resolve('成功') // 1秒后输出 失败
    }, 1000)
}).then(res => console.log(res), err => console.log(err))

```

js 复制代码



897



98



收藏

then支持 链式调用，下一次then执行 受上一次then返回值的影响，给大家举个例子：

```
// 链式调用 输出 200
const p3 = new Promise((resolve, reject) => {
  resolve(100)
}).then(res => 2 * res, err => console.log(err))
  .then(res => console.log(res), err => console.log(err))

// 链式调用 输出300
const p4 = new Promise((resolve, reject) => {
  resolve(100)
}).then(res => new Promise((resolve, reject) => resolve(3 * res)), err => console.log(err))
  .then(res => console.log(res), err => console.log(err))
```

js 复制代码

从上方例子，我们可以获取到几个知识点：

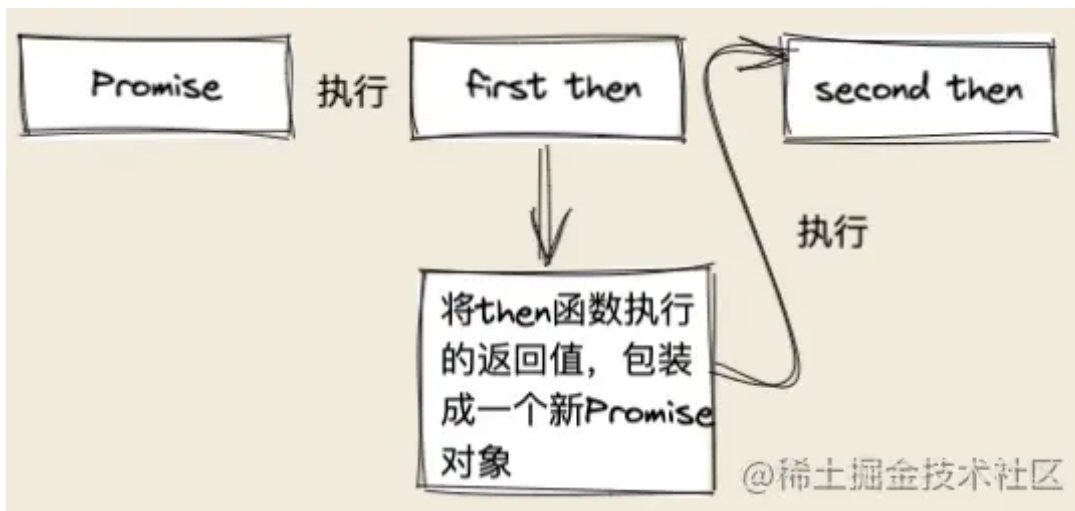
- 1、then方法本身会返回一个新的Promise对象
- 2、如果返回值是promise对象，返回值为成功，新promise就是成功
- 3、如果返回值是promise对象，返回值为失败，新promise就是失败
- 4、如果返回值非promise对象，新promise对象就是成功，值为此返回值

咱们知道then是Promise上的方法，那如何实现then完还能再then呢？很简单，then执行后返回一个 Promise对象 就行了，就能保证then完还能继续执行then：

 897

 98

 收藏



代码实现:

js 复制代码

```
then(onFulfilled, onRejected) {  
  // 接收两个回调 onFulfilled, onRejected  
  
  // 参数校验, 确保一定是函数  
  onFulfilled = typeof onFulfilled === 'function' ? onFulfilled : val => val  
  onRejected = typeof onRejected === 'function' ? onRejected : reason => { throw reason }  
  
  var thenPromise = new MyPromise((resolve, reject) => {  
  
    const resolvePromise = cb => {  
      try {  
        const x = cb(this.PromiseResult)  
        if (x === thenPromise) {
```

👍 897

💬 98

★ 收藏

```

    }
    if (x instanceof MyPromise) {
        // 如果返回值是Promise
        // 如果返回值是promise对象, 返回值为成功, 新promise就是成功
        // 如果返回值是promise对象, 返回值为失败, 新promise就是失败
        // 谁知道返回的promise是失败成功? 只有then知道
        x.then(resolve, reject)
    } else {
        // 非Promise就直接成功
        resolve(x)
    }
} catch (err) {
    // 处理报错
    reject(err)
    throw new Error(err)
}
}

if (this.PromiseState === 'fulfilled') {
    // 如果当前为成功状态, 执行第一个回调
    resolvePromise(onFulfilled)
} else if (this.PromiseState === 'rejected') {
    // 如果当前为失败状态, 执行第二个回调
    resolvePromise(onRejected)
} else if (this.PromiseState === 'pending') {
    // 如果状态为待定状态, 暂时保存两个回调
    // 如果状态为待定状态, 暂时保存两个回调
    this.onFulfilledCallbacks.push(resolvePromise.bind(this, onFulfilled))
    this.onRejectedCallbacks.push(resolvePromise.bind(this, onRejected))
}
}

```



```
// 返回这个包装的Promise
return thenPromise

}
```

现在大家可以试试效果怎么样了，大家要[边敲边试](#)哦：

```
const test3 = new Promise((resolve, reject) => {
  resolve(100) // 输出 状态:成功 值: 200
  // reject(100) // 输出 状态:成功 值:300
}).then(res => 2 * res, err => 3 * err)
  .then(res => console.log('成功', res), err => console.log('失败', err))

const test4 = new Promise((resolve, reject) => {
  resolve(100) // 输出 状态:失败 值:200
  // reject(100) // 输出 状态:成功 值:300
  // 这里可没搞反哦。真的搞懂了,就知道了为啥这里是反的
}).then(res => new Promise((resolve, reject) => reject(2 * res)), err => new Promise((resolve, reject) => resolve(3 * err)))
  .then(res => console.log('成功', res), err => console.log('失败', err))
```

js 复制代码

4. 微任务

看过 [js执行机制](#) 的兄弟都知道，then方法是 [微任务](#)，啥叫微任务呢？其实不知道也不要紧，我通过下面例子让你知道：

js 复制代码



```
}).then(res => console.log(res), err => console.log(err))
```

```
console.log(2)
```

输出顺序是 2 1

为啥不是 1 2 呢？因为then是个微任务啊。。。同样，我们也要给我们的MyPromise加上这个特性(我这里使用定时器，大家别介意哈)

只需要让 `resolvePromise` 函数 异步执行就可以了

```
const resolvePromise = cb => {
  setTimeout(() => {
    try {
      const x = cb(this.PromiseResult)
      if (x === thenPromise) {
        // 不能返回自身哦
        throw new Error('不能返回自身。。。')
      }
      if (x instanceof MyPromise) {
        // 如果返回值是Promise
        // 如果返回值是promise对象，返回值为成功，新promise就是成功
        // 如果返回值是promise对象，返回值为失败，新promise就是失败
        // 谁知道返回的promise是失败成功？只有then知道
        x.then(resolve, reject)
      } else {
        // 非Promise就直接成功
        resolve(x)
      }
    } catch (err) {
```

js 复制代码



897



98



收藏

```
        reject(err)
        throw new Error(err)
    }
})
}
```

看看效果：

```
const test4 = new MyPromise((resolve, reject) => {
  resolve(1)
}).then(res => console.log(res), err => console.log(err))
```

```
console.log(2)
```

输出顺序 2 1

js 复制代码

其他方法

这些方法都比较简单，我就不太过详细地讲了，大家也可以借这个机会，自己摸索，巩固这篇文章的知识。

all

- 接收一个Promise数组，数组中如有非Promise项，则此项当做成功

 897

 98

 收藏

- 如果有一个Promise失败，则返回这个失败结果

js 复制代码

```
static all(promises) {
  const result = []
  let count = 0
  return new MyPromise((resolve, reject) => {
    const addData = (index, value) => {
      result[index] = value
      count++
      if (count === promises.length) resolve(result)
    }
    promises.forEach((promise, index) => {
      if (promise instanceof MyPromise) {
        promise.then(res => {
          addData(index, res)
        }, err => reject(err))
      } else {
        addData(index, promise)
      }
    })
  })
}
```

race

- 接收一个Promise数组，数组中如有非Promise项，则此项当做成功




```
static race(promises) {  
  return new MyPromise((resolve, reject) => {  
    promises.forEach(promise => {  
      if (promise instanceof MyPromise) {  
        promise.then(res => {  
          resolve(res)  
        }, err => {  
          reject(err)  
        })  
      } else {  
        resolve(promise)  
      }  
    })  
  })  
}
```

allSettled

- 接收一个Promise数组，数组中如有非Promise项，则此项当做成功
- 把每一个Promise的结果，集成数组，返回

```
static allSettled(promises) {  
  return new Promise((resolve, reject) => {  
    const res = []  
    let count = 0
```



897



98



收藏

```
        status,
        value
      }
      count++
      if (count === promises.length) {
        resolve(res)
      }
    }
    promises.forEach((promise, i) => {
      if (promise instanceof MyPromise) {
        promise.then(res => {
          addData('fulfilled', res, i)
        }, err => {
          addData('rejected', err, i)
        })
      } else {
        addData('fulfilled', promise, i)
      }
    })
  })
}
```

any

any与all相反

- 接收一个Promise数组，数组中如有非Promise项，则此项当做成功



- 如果所有Promise都失败，则报错

js 复制代码

```
static any(promises) {  
  return new Promise((resolve, reject) => {  
    let count = 0  
    promises.forEach((promise) => {  
      promise.then(val => {  
        resolve(val)  
      }, err => {  
        count++  
        if (count === promises.length) {  
          reject(new AggregateError('All promises were rejected'))  
        }  
      })  
    })  
  })  
}
```

结语

再也不怕面试官问你Promise原理啦哈哈哈哈 😄

如果你觉得此文对你有一丁点帮助，点个赞，鼓励一下林三心哈哈。

[如果你想一起学习前端或者塔角](#)，[那你可以加我](#)，[加入我的塔角学习群](#)，[点击这里](#) ---> [塔角进阶](#)



897



98



收藏

如果你是有其他目的的，别加我，我不想跟你交朋友，我只想简简单单学习前端，不想搞一些有的没的!!!

分类： 前端

标签： 前端

JavaScript

文章被收录于专栏：



面试——百毒不侵

面试

关注专栏



代码人生

代码过程的一些趣事

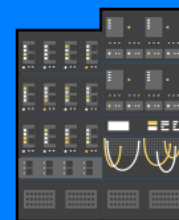
关注专栏

 897

 98

 收藏

找对——
属于你的
技术圈子



回复进群加入
掘金
微信交流群



评论

输入评论 (Enter换行, Ctrl + Enter发送)

热门评论

 897

 98

 收藏

大佬，好像用定时器后，
test4就不能输出值了，这是为啥

👍 点赞 💬 5

 **GeorgeSmith** Lv1


7月前

就是这个

```
const test4 = new MyPromise((resolve, reject) => {  
  resolve(100) // 输出 状态：失败 值：200  
  // reject(100) // 输出 状态：成功 值：300  
  // 这里可没搞反哦。真的搞懂了，就知道了为啥这里是反的  
}).then(res => new MyPromise((resolve, reject) => reject(2 * res)), err => new MyPromise((resolve, reject) => resolve(2 * res)))...
```

[展开](#)

👍 点赞 💬 回复

 **Sunshine_Lin** Lv6 (作者)

7月前

这里我写错了，你改一下应该就好了，抱歉哈

👍 1 💬 回复

[查看更多回复](#) ▾

👍 897

💬 98

☆ 收藏

```
if (x === thenPromise) {  
  // 不能返回自身哦  
  throw new Error('不能返回自身。。。')  
}
```

我像问一下，x在什么情况下才会全等于thenPromise，x和thenPromise不应该是两个完全不同的对象嘛？

👍 点赞 💬 4



Sunshine_Lin Lv6 (作者)

3月前

then返回自己？

👍 点赞 💬 回复



yoker 回复 **Sunshine_Lin**

3月前

还是不太明白，x是cb函数的返回值，thenPromise是新创建的MyPromise实例对象。
如何才能使cb函数的返回值 全等于 一个完全新创建出来的yPromise实例对象。

“then返回自己？”

👍 点赞 💬 回复

查看更多回复 ▾

全部评论 98

🕒 最新

🔥 最热



SuperFan_小武 Lv1 前端开发

10天前

👍 897

💬 98

★ 收藏

👍 点赞 💬 回复

静听花亦落 老年前端 @ 阿巴阿巴

12天前

文章说用数组保存回调，是因为promise实例会有多次then，这个地方不太理解，什么情况下会有多次then的情况呢？
链式调用应该不算多次调用吗，毕竟每次then都会返回一个新的promise
有没有大佬举个例子啊

👍 点赞 💬 回复

| 雨陌 前端 @ 小码农...

19天前

var thenPromise 改成 用let或者const声明时，then方法链式调用，两个then方法只会输出第一个then方法的值是为啥

👍 点赞 💬 回复

李强56

25天前

throw 章节里的 try catch 应该加到哪里？

👍 1 💬 回复

用户8835717137... Lv1

27天前

牛，三心哥

👍 点赞 💬 回复

HY小菜鸡 Lv1

1月前

while(this.onFulfilledCallbacks.length) { this.onFulfilledCallbacks.shift()(this.PromiseResult)} 这里有问题吧？

👍 897

💬 98

☆ 收藏

还会执行吧？？

👍 点赞 💬 回复

OrzR3 Lv2

1月前

先码再看

👍 点赞 💬 回复

zxp123

1月前

😂 文章中特意提示的不是搞反了地方，值错了，`resolve(100)` // 输出 状态：失败 值：200，这里蛮误导人了，建议早点改一下，原本一眼可以看出的 特意说了些 反而还得多看几眼

👍 点赞 💬 1

Sunshine_Lin Lv6 (作者)

1月前

修改了。。尴尬

👍 点赞 💬 回复

MrShudong Lv1

2月前

发现一个问题 这样报错的话 执行到cb就报错了 麻烦作者大大检查瞅瞅 😁

👍 897

💬 98

★ 收藏

谦Q_Q

2月前

应该是失败200 成功300吧？ 感觉写反了呢

👍 点赞 💬 回复

early_birds Lv1 前端 @ ? ? ?

2月前

executor()是干嘛的？ 执行代码？ 为什么不直接写this.resolve() this.reject()?

👍 点赞 💬 3

Sunshine_Lin Lv6 (作者)

2月前

executor传进来的函数呀。。 不写this.resolve是因为这样会有this指向问题

👍 1 💬 回复

early_birds Lv1 回复 Sunshine_Lin

2月前

哦哦哦，没看到形参，起名好高级，我以为这是个我没见过的什么函数，百度搜了半天😓😓

“executor传进来的函数呀。。 不写this.resolve是因为这样会有this指向问题”

👍 点赞 💬 回复

查看更多回复 ▾

yoker

3月前

if (x === thenPromise) {

👍 897

💬 98

☆ 收藏

```
throw new Error('不能返回自身。。。')
}
```

我像问一下，x在什么情况下才会全等于thenPromise，x 和thenPromise不应该是两个完全不同的对象嘛？

👍 点赞 💬 4



Sunshine_Lin Lv6 (作者)

3月前

then返回自己？

👍 点赞 💬 回复



yoker 回复 **Sunshine_Lin**

3月前

还是不太明白，x是cb函数的返回值，thenPromise是新创建的MyPromise实例对象。
如何才能成为cb函数的返回值 全等于 一个完全新建出来的yPromise实例对象。

“then返回自己？”

👍 点赞 💬 回复

查看更多回复 ▾



青春的前小腿 菜鸡前端

3月前

请问用setTimeout好像是变成了下一个宏任务吧...不是微任务啊

👍 点赞 💬 4



Sunshine Lin Lv6 (作者)

3月前

👍 897

💬 98

☆ 收藏

👍 点赞 💬 回复

 青春的前小腿 回复 **Sunshine_Lin**

3月前

原来如此

“文中说了，这里使用setTimeout代替微任务。”

👍 点赞 💬 回复


查看更多回复 ▾

 **zwj**

3月前

大佬，这里有问题吧

👍 点赞 💬 2

 **Sunshine_Lin** Lv6 (作者)

3月前

啥问题

👍 点赞 💬 回复

 **zwj** 回复 **Sunshine_Lin**

3月前

眼拙，看错了哈哈

👍 897

💬 98

★ 收藏

👍 点赞 💬 回复

user1738467305...

3月前

很详细哦

👍 点赞 💬 1

Sunshine_Lin Lv6 (作者)

3月前

哈哈

👍 点赞 💬 回复

KrisW

3月前

牛逼!!!

👍 点赞 💬 回复

海绵宝宝2号 前端挖掘机 @ 潜心

3月前

我看大家基本上都是settimeout模拟的异步，请问promise源码里的微任务怎么实现的呢，js没办法模拟吗

👍 点赞 💬 1

s_qin

3月前

queueMicrotask

👍 897

💬 98

★ 收藏

user8280318367...

3月前

initBind 这个方法可以省略

改写resolve = (value) =>{...} reject = (reasons) =>{...}

zh.javascript.info

👍 点赞 💬 回复

XW429 前端开发搞笑师 @ 广州...

4月前

console.log('p',p)其实可以写哦console.log({p})

👍 点赞 💬 回复

深圳矿工 Lv1 前端开发工程师 @ 平安...

4月前

3.链式调用第一个代码块种，resolve(100)用定时器包裹，就无效了

👍 1 💬 回复

查看全部 98 条回复 ▾

相关推荐

myounchina | 7月前 | 前端

👍 897

💬 98

☆ 收藏

👁 1760 👍 35 💬 8

圆圆01 | 3月前 | JavaScript · 前端

手把手一行一行代码教你“手写Promise”，完美通过 Promises/A+ 官方872个测试用例

👁 3.3w 👍 620 💬 184

摸鱼的春哥 | 2月前 | 前端 · JavaScript

2022，前端的天☁️要怎么变？

👁 6.8w 👍 665 💬 250

HighClassLickDog | 4月前 | 前端 · JavaScript

因为懒，我把公司的后管定制成了低代码中台

👁 6.4w 👍 459 💬 171

前端胖头鱼 | 3月前 | 前端 · JavaScript · 面试

因为实现不了Promise.all，一场面试凉凉了

👁 4.4w 👍 603 💬 96

CUGGZ | 5月前 | 前端 · JavaScript · 程序员

33个非常实用的JavaScript一行代码，建议收藏！

👁 7.3w 👍 1730 💬 59

前端胖头鱼 | 5月前 | JavaScript · Vue.js · 前端

👍 897

💬 98

☆ 收藏

👁 7.6w 👍 1067 💬 233

Sunshine_Lin | 7月前 | 前端 · JavaScript · Vue.js

15张图，20分钟吃透Diff算法核心原理，我说的！！

👁 5.7w 👍 2232 💬 246

大帅老猿 | 9月前 | 前端 · JavaScript · HTML

2天赚了4个W，手把手教你用Threejs搭建一个Web3D汽车展厅！

👁 5.9w 👍 1420 💬 240

大海我来了 | 1年前 | JavaScript

死磕 36 个 JS 手写题（搞懂后，提升真的大）

👁 10.3w 👍 3384 💬 185

Sunshine_Lin | 5月前 | 前端 · JavaScript · ECMAScript 6

「万字总结」熬夜总结50个JS的高级知识点，全都会你就是神！！

👁 7.5w 👍 2179 💬 107

浪里行舟 | 3年前 | JavaScript · 前端

九种跨域方式实现原理（完整版）

👁 13.6w 👍 2613 💬 107

前端阿飞 | 4月前 | 前端 · JavaScript

👍 897

💬 98

☆ 收藏

👁 8.9w 👍 2222 💬 178

煜成 | 9月前 | 前端

手写Promise.all和Promise.race

👁 1472 👍 6 💬 1

蔓蔓雒轩 | 3年前 | JavaScript · 前端 · Promise

Promise不会?? 看这里!!! 史上最通俗易懂的Promise!!!

👁 7.8w 👍 1499 💬 42

大帅老猿 | 10月前 | 前端 · JavaScript

产品经理:你能不能用div给我画条龙?

👁 10.8w 👍 2907 💬 576

lzg9527 | 2年前 | 前端 · JavaScript

总结移动端H5开发常用技巧(干货满满哦!)

👁 5.4w 👍 1925 💬 58

阿里南京技术专刊 | 3年前 | Angular.js · Git · JavaScript

优雅的提交你的 Git Commit Message

👁 9.6w 👍 1213 💬 39

程序员依扬 | 2年前 | 面试 · 前端

👍 897

💬 98

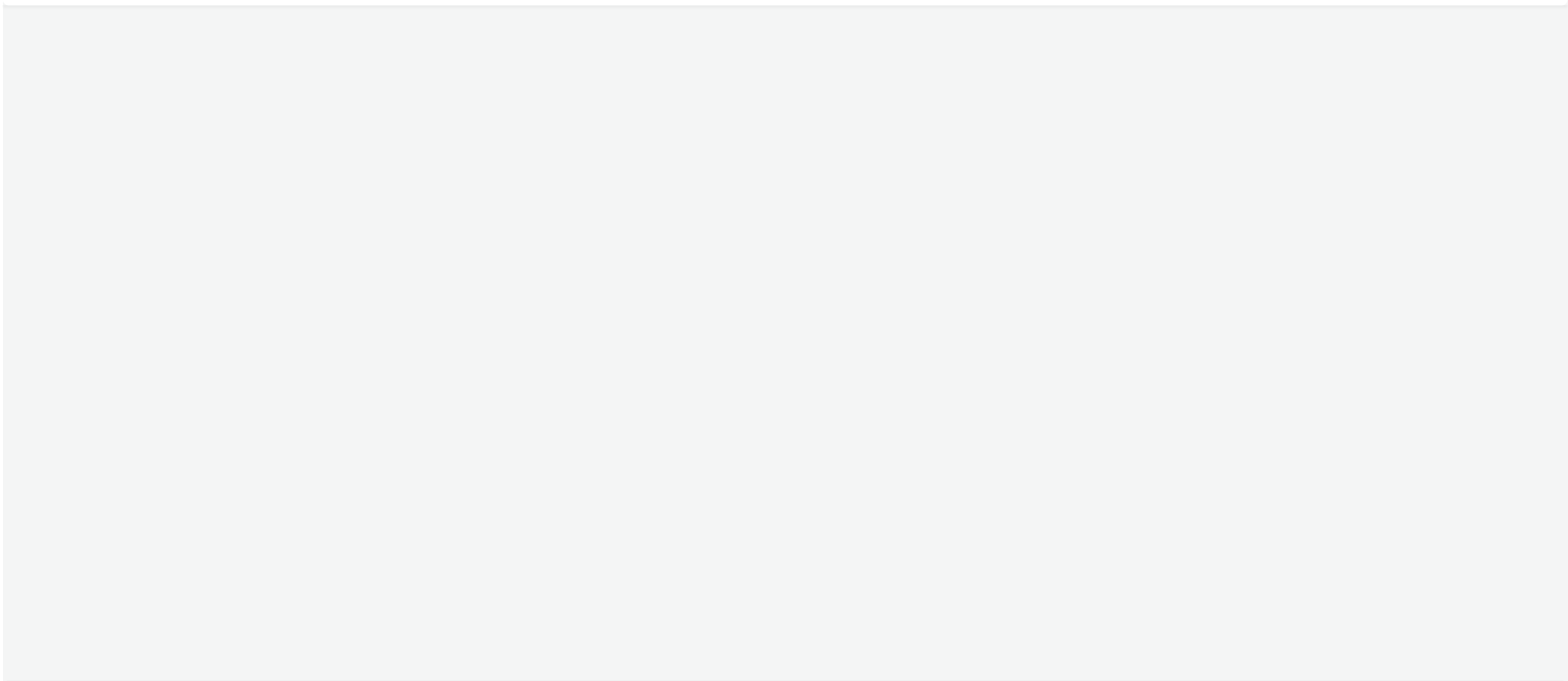
☆ 收藏

👁 52.7w 👍 9279 💬 308

非优秀程序员 | 4月前 | 前端 · JavaScript

如何用 CSS 中写出超级美丽的阴影效果

👁 31.3w 👍 235 💬 12



👍 897

💬 98

☆ 收藏