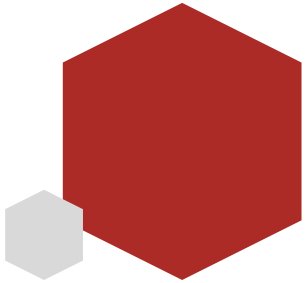
手写Promise

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌



手写Promise

手写Promise

需求:手写Promise,实现所有功能,并通过Promise/A+测试


```
const p = new HMPromise((resolve, reject) => {
  setTimeout(() => {
    resolve('success')
    // reject('error')
  }, 1000);
})

p.then(res => {
  console.log('res:', res)
  return 'success2'
}, err => {
  console.log('err:', err)
}).then(res2 => {
  console.log('res2:', res2)
})
```

```
// 实例方法
.catch(...)
.finally(...)

// 静态方法
.resolve(...)
.reject(...)
.race(...)
.all(...)
.allSettled(...)
.any(...)
```

```
872 passing
```

核心功能

- 构造函数
- 状态及原因
- then方法
- 异步任务
- 链式编程

```
const p = new HMPromise((resolve, reject) => {
  setTimeout(() => {
    resolve('success')
    // reject('error')
  }, 1000);
})

p.then(res => {
  console.log('res:', res)
  return 'success2'
}, err => {
  console.log('err:', err)
}).then(res2 => {
  console.log('res2:', res2)
})
```

核心功能

核心步骤： 定义类 → 添加构造函数 → 定义resolve/reject → 执行回调函数

- 构造函数
- 状态及原因
- then方法
- 异步任务
- 链式编程

```
const p = new HMPromise((resolve, reject) => {
  resolve('success')
  // reject('error')
})
```

核心功能

核心步骤： 添加状态 → 添加原因 → 调整resolve/reject → 状态不可逆

- 构造函数
- 状态及原因
- then方法
- 异步任务
- 链式编程

```
const p = new HMPromise((resolve, reject) => {
  resolve('success') // pending -> fulfilled
  // reject('error') // pending -> rejected
})
p.state // 状态
p.result // 原因
```

核心功能

核心步骤：

- 构造函数
- 状态及原因
- then方法
  - 成功和失败回调
  - 异步和多次调用
- 异步任务
- 链式编程

```
const p = new HMPromise((resolve, reject) => {
    resolve('success')
    // reject('error')
})
p.then(res => {
  console.log('成功回调:', res)
}, err => {
  console.log('失败回调:', err)
})
```

核心功能

核心步骤： 定义实例属性 → 保存回调函数 → 调用成功回调 / 调用失败回调

- 构造函数
- 状态及原因
- then方法
  - 成功和失败回调
  - 异步和多次调用
- 异步任务
- 链式编程

```
const p = new HMPromise((resolve, reject) => {
  setTimeout(() => {
    resolve('success')
    // reject('error')
  }, 2000);
})

p.then(res => {
  console.log('then1:', res)
}, err => {
  console.log('then1:', err)
})

p.then(res => {
  console.log('then2:', res)
}, err => {
  console.log('then2:', err)
})
```

核心功能

需求: 输出 top bottom success

- 构造函数
- 状态及原因
- then方法
- 异步任务
- 链式编程

```
console.log('top')
const p = new HMPromise((resolve, reject) => {
  resolve('success')
})
p.then(res => {
  console.log(res)
})
console.log('bottom')
```

核心功能

- 构造函数
- 状态及原因
- then方法
- 异步任务
  - 核心api
  - 函数封装
- 链式编程

核心api:

1. **vue2:** Promise.then、MutationObserver 、 setImmediate 、 setTimeout
2. **选用:** queueMicrotask 、MutationObserver 、 setTimeout

```
queueMicrotask(() => {
  // ....
})
```

```
const obs = new MutationObserver(() => {
  //...
})
const divNode = document.createElement('div')
obs.observe(divNode, { childList: true })
divNode.innerText = 'itheima 666'
```

核心功能

核心步骤： 定义函数 → 调用核心api → 调用函数

- 构造函数
- 状态及原因
- then方法
- 异步任务
  - 核心api
  - 函数封装
- 链式编程

```
console.log('top')
const p = new HMPromise((resolve, reject) => {
  resolve('success')
})
p.then(res => {
  console.log(res)
})
console.log('bottom')
```

核心功能

核心步骤： 返回新Promise实例 → 获取返回值 → 处理返回值 / 处理异常 / 处理返回Promise / 处理重复引用

- 构造函数
- 状态及原因
- then方法
- 异步任务
- 链式编程 → fulfilled状态 / rejected状态 / pending状态

```
const p = new HMPromise((resolve, reject) => {
  resolve(1)
})
p.then(res => {
  console.log(res)
  // throw 'throw-error'
  return 2
}).then(res => {
  console.log(res)
}, err => {
  console.log(err)
})
```

核心功能

核心步骤：

返回新Promise实例 → 获取返回值 → 处理返回值 / 处理异常 / 处理返回Promise → 调用then方法 / 处理重复引用

- 构造函数
- 状态及原因
- then方法
- 异步任务
- 链式编程
  - fulfilled状态
  - rejected状态
  - pending状态

```
const p = new HMPromise((resolve, reject) => {
  resolve(1)
})
p.then(res => {
  return new HMPromise((resolve, reject) => {
    resolve(2)
    // reject('error')
  })
}).then(res => {
  console.log('p2:', res) // 2
}, err => {
  console.log('p2:', err) // error
})
```

核心功能

核心步骤： 返回新Promise实例 → 获取返回值

- 处理返回值
- 处理异常
- 处理返回Promise
- 处理重复引用 → 抛出异常

```
throw new TypeError('')
```

- 构造函数
- 状态及原因
- then方法
- 异步任务
- 链式编程
  - fulfilled状态
  - rejected状态
  - pending状态

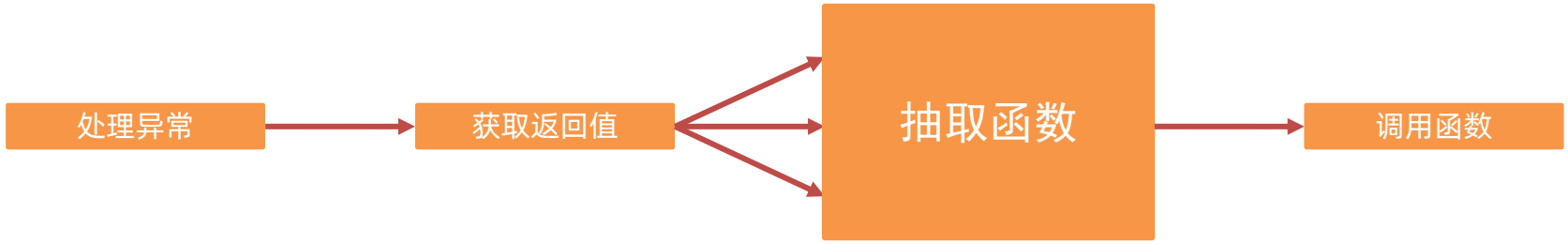
```
const p = new HMPromise((resolve, reject) => {
  resolve(1)
})
const p2 = p.then(res => {
  return p2
})

p2.then(
  res => { },
  err => console.log('err:', err))
```

```
▶Uncaught (in promise) TypeError: Chaining cycle detected for promise #<Promise>
```

核心功能

核心步骤：



- 构造函数
- 状态及原因
- then方法
- 异步任务
- 链式编程
  - fulfilled状态
  - rejected状态
  - pending状态

```
const p = new HMPromise((resolve, reject) => {
  reject(1)
})
const p2 = p.then(undefined, err => {
  throw 'error'
  // return p2
  // return 2
  // return new HMPromise((resolve, reject) =>
{
  //   resolve('HMPromise-2')
  // })
})
p2.then(res => {
  console.log('p2-res:', res)
}, err => {
  console.log('p2-err:', err)
})
```

核心功能

核心步骤： 处理异常 → 获取返回值 → 调用函数

- 构造函数
- 状态及原因
- then方法
- 异步任务
- 链式编程
  - fulfilled状态
  - rejected状态
  - pending状态

```
const p = new HMPromise((resolve, reject) => {
  setTimeout(() => {
    resolve(1)
  }, 2000)
})
const p2 = p.then(res => {
  throw 'error'
  // return p2
  // return 2
  // return new HMPromise((resolve, reject) => {
  //   resolve('resolve-2')
  //   // reject('reject-2')
  // })
})

p2.then(res => {
  console.log('p2-res:', res)
}, err => {
  console.log('p2-err:', err)
})
```

手写Promise

需求:手写Promise,实现所有功能,并通过Promise/A+测试


```
const p = new HMPromise((resolve, reject) => {
  setTimeout(() => {
    resolve('success')
    // reject('error')
  }, 1000);
})

p.then(res => {
  console.log('res:', res)
  return 'success2'
}, err => {
  console.log('err:', err)
}).then(res2 => {
  console.log('res2:', res2)
})
```

```
// 实例方法
.catch(...)
.finally(...)

// 静态方法
.resolve(...)
.reject(...)
.race(...)
.all(...)
.allSettled(...)
.any(...)
```

```
872 passing
```

实例方法

核心步骤： 参考文档 → 内部调用then → 处理异常

catch

finally

```
const p = new HMPromise((resolve, reject) => {
  reject('reject-error')
  // throw 'throw-error'
})

p.then(res => {
  console.log('res:', res)
}).catch(err => {
  console.log('err:', err)
})
```

实例方法

核心步骤： 参考文档 → 内部调用then

catch

finally

```
const p = new HMPromise((resolve, reject) => {
  // resolve('resolve-res')
  // reject('reject-error')
  // throw 'throw-error'
})

p.then(res => {
  console.log('res:', res)
}).catch(err => {
  console.log('err:', err)
}).finally(() => {
  console.log('finally')
})
```

手写Promise

需求:手写Promise,实现所有功能,并通过Promise/A+测试


```
const p = new HMPromise((resolve, reject) => {
  setTimeout(() => {
    resolve('success')
    // reject('error')
  }, 1000);
})

p.then(res => {
  console.log('res:', res)
  return 'success2'
}, err => {
  console.log('err:', err)
}).then(res2 => {
  console.log('res2:', res2)
})
```

```
// 实例方法
.catch(...)
.finally(...)

// 静态方法
.resolve(...)
.reject(...)
.race(...)
.all(...)
.allSettled(...)
.any(...)
```

```
872 passing
```

静态方法

核心步骤：

- resolve
- reject
- race
- all
- allSettled
- any

```
HMPromise.resolve(new HMPromise((resolve, reject) => {
  // resolve('resolve')
  // reject('reject')
  // throw 'error'
})).then(res => {
  console.log('res:', res)
}, err => {
  console.log('err:', err)
})

HMPromise.resolve('itheima').then(res => {
  console.log(res)
})
```

静态方法

核心步骤： 参考文档 → 返回rejected状态的Promise

- resolve
- reject
- race
- all
- allSettled
- any

```
HMPromise.reject('error').catch(res => {
  console.log(res)
})
```

静态方法

核心步骤： 参考文档 → 返回Promise → 判断是否为数组 → 等待第一个敲定

- resolve
- reject
- race
- all
- allSettled
- any

```
const p1 = new HMPromise((resolve, reject) => {
  setTimeout(() => {
    resolve(1)
  }, 2000)
})
const p2 = new HMPromise((resolve, reject) => {
  setTimeout(() => {
    reject(2)
  }, 1000)
})

HMPromise.race([p1, p2, 'itheima']).then((res) => {
  console.log('res:', res)
}, err => {
  console.log('err:', err)
})
```

```
TypeError: undefined is not iterable (cannot read property Symbol(Symbol.iterator))
```

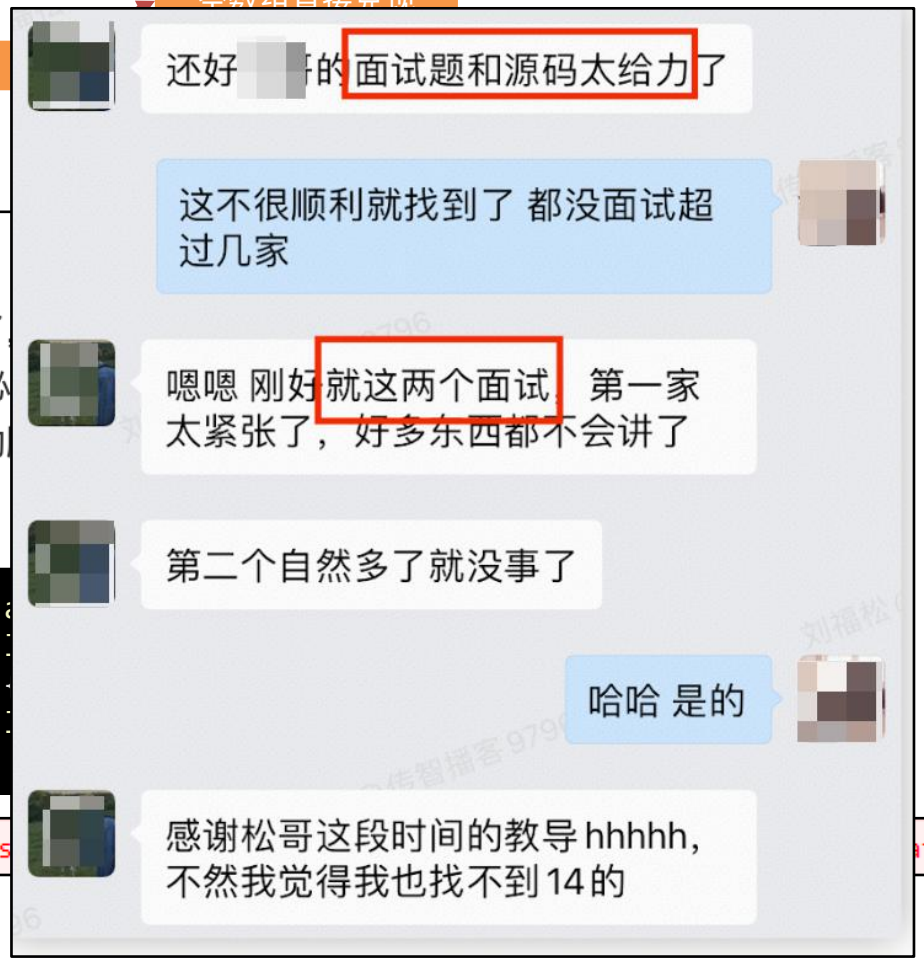
静态方法

核心步骤： 参考文档 → 返回Promise → 判断是否为数组

- resolve
- reject
- race
- all
- allSettled
- any

1.自我介绍
2.Websocket 和Websocket长连接
3.节流防抖
4.Http 缓存
5.es6新特性
6.垃圾内存机制
7.闭包使用和回收
8.Promise 如何解决回调地狱➕all的底层原理
9.本地缓存
10.小程序打包过大，优化
11.路由
12.vue 生命周期
13.权限管理

空数组直接总现



静态方法

核心步骤：

- resolve
- reject
- race
- all
- allSettled
- any

```
const results = []
let count = 0
promises.forEach((p, index) => {
  HMPromise.resolve(p).then(
    res => {
      results[index] = res
      count++
      count === promises.length && resolve(results)
    },
    err => {
      reject(err)
    })
})
```

```
const p1 = HMPromise.resolve(1)
const p2 = new HMPromise((resolve, reject) =>
{
  setTimeout(() => {
    resolve(2)
  }, 1000)
})
const p3 = 3

[p1, p2, p3]
```

```
[ ,  , 3]
```

静态方法

核心步骤： 参考文档

- resolve
- reject
- race
- all
- allSettled
  - 核心用法
  - 手写实现
- any

静态方法

核心步骤： 参考文档 → 返回Promise → 判断是否为数组 → 空数组直接兑现 / 等待全部敲定 → 记录结果 → 处理兑现 / 处理拒绝

- resolve
- reject
- race
- all
- allSettled
  - 核心用法
  - 手写实现
- any

```
const p1 = HMPromise.resolve(1)
const p2 = 2
const p3 = new HMPromise((resolve, reject) => {
  setTimeout(() => {
    reject(3)
  }, 1000)
})
HMPromise.allSettled([p1, p2, p3]).then(res => {
  console.log('res:', res)
}, err => {
  console.log('err:', err)
})
```

```
▶ 0: {status: 'fulfilled', value: 1}
▶ 1: {status: 'fulfilled', value: 2}
▶ 2: {status: 'rejected', reason: 3}
  length: 3
```

```
TypeError: undefined is not iterable (cannot read property Symbol(Symbol.iterator))
```

静态方法

核心步骤： 参考文档

- resolve
- reject
- race
- all
- allSettled
- any
  - 核心用法
  - 手写实现

静态方法

核心步骤： 参考文档 → 返回Promise,数组判断 → 空数组直接拒绝 / 等待结果 → 第一个兑现 / 全部拒绝

- resolve
- reject
- race
- all
- allSettled
- any → 核心用法 / 手写实现

```
AggregateError: All promises were rejected
▶ errors: (3) [1, 2, 3]
  message: "All promises were rejected"
  stack: "AggregateError: All promises were rejected"
```

```
const p1 = new HMPromise((resolve, reject) => {
  setTimeout(() => {
    reject(1)
  }, 2000)
})
const p2 = 2
const p3 = new HMPromise((resolve, reject) => {
  setTimeout(() => {
    resolve(3)
    // reject(3)
  }, 1000)
})

HMPromise.any([p1, p2, p3]).then(res => {
  console.log('res:', res)
}, err => {
  console.dir(err)
})
```

```
TypeError: undefined is not iterable (cannot read property Symbol(Symbol.iterator))
```

手写Promise

需求:手写Promise,实现所有功能,并通过Promise/A+测试


```
const p = new HMPromise((resolve, reject) => {
  setTimeout(() => {
    resolve('success')
    // reject('error')
  }, 1000);
})

p.then(res => {
  console.log('res:', res)
  return 'success2'
}, err => {
  console.log('err:', err)
}).then(res2 => {
  console.log('res2:', res2)
})
```

```
// 实例方法
.catch(...)
.finally(...)

// 静态方法
.resolve(...)
.reject(...)
.race(...)
.all(...)
.allSettled(...)
.any(...)
```

```
872 passing
```

Promise/A+

Promise/A+规范是由社区提出的，最早的Promise是由社区首先提出和实现的

社区提出并实现

1. Q
2. when
3. WinJS
4. RSVP.js

...

ES6加入语言标准

Promise/A+

通过 promises-aplus-tests 测试代码是否符合Promise/A+规范

核心步骤：

使用CommonJS暴露对象 → 下包 → 配置并执行命令

1. 提供deferred方法,返回对象{promise,resolve,reject}
   1.1 promise: pending状态的promise实例
   1.2 resolve: 以传入的原因兑现promise
   1.3 reject: 以传入的原因拒绝promise

1. 初始化项目: npm init -y
2. 下包: npm i promises-aplus-tests -D

命令:promises-aplus-tests 代码文件

```
module.exports = {
  deferred() {
    const res = {}
    res.promise = new HMPromise((resolve, reject) => {
      res.resolve = resolve
      res.reject = reject
    })
    return res
  }
}
```

传智教育旗下高端IT教育品牌