25. 手写一个简化版 Promise

1. 核心概念 (Core Concept)

Promise 是 JavaScript 中用于处理异步操作的一种机制，它代表一个异步操作的最终完成或失败，以及其结果值。手写一个简化版 Promise 的目标是理解其核心状态管理（Pending, Fulfilled, Rejected）和状态转换过程，以及如何通过 `then` 方法处理异步结果。

2. 为什么需要它？ (The "Why")

1. **解决回调地狱 (Callback Hell):** Promise 提供了一种比传统嵌套回调更清晰、更易于管理异步流程的方式。
2. **统一异步操作的写法:** 无论是定时器、网络请求还是其他异步事件，都可以封装成 Promise，使用统一的 `.then().catch()` 链式调用风格。
3. **更容易处理错误:** `.catch()` 提供了一种集中处理异步操作链中任意环节错误的方式，避免了在每个回调函数中重复进行错误检查。

3. API 与用法 (API & Usage)

手写一个简化版 Promise 的核心在于实现 Promise 的构造函数 (`Promise`) 和其原型上的 `then` 方法。

简化版 Promise 的核心实现思路:

- **状态 (State):** 维护一个内部状态，有三种：`'pending'` （待定）、`'fulfilled'` （已兑现/成功）和 `'rejected'` （已拒绝/失败）。
- **结果值 (Value/Reason):** 当状态变为 fulfilled 时，记录成功的值 (value)；当状态变为 rejected 时，记录失败的原因 (reason)。
- **回调函数队列 (Callbacks):** 在 trạng thái 仍是 `'pending'` 时，存储通过 `.then` 方法注册的成功和失败回调函数。
- **resolve/reject 方法:** 由 Promise 构造函数的 executor 函数提供，用于改变 Promise 的状态并触发相应的回调函数。
- **then 方法:** 注册成功和失败的回调函数。如果在调用 `then` 时 Promise 已处于 fulfilled 或 rejected 状态，则立即执行相应的回调；否则，将回调添加到队列中。`then` 方法必须返回一个新的 Promise，以支持链式调用。

核心代码结构 (简化版):

```
// 伪代码，演示核心结构
class SimplePromise {
    constructor(executor) {
        this.state = 'pending'; // 'pending', 'fulfilled', 'rejected'
        this.value = undefined;
```

```
        this.reason = undefined;
        this.onFulfilledCallbacks = []; // 存储成功回调
        this.onRejectedCallbacks = []; // 存储失败回调

        const resolve = (value) => {
            if (this.state === 'pending') {
                this.state = 'fulfilled';
                this.value = value;
                // 状态确定后，执行所有存储的成功回调
                this.onFulfilledCallbacks.forEach(callback => {
                    // 为了简化，这里不考虑回调返回Promise的情况
                    callback(this.value);
                });
            }
        };

        const reject = (reason) => {
            if (this.state === 'pending') {
                this.state = 'rejected';
                this.reason = reason;
                // 状态确定后，执行所有存储的失败回调
                this.onRejectedCallbacks.forEach(callback => {
                    // 为了简化，这里不考虑回调返回Promise的情况
                    callback(this.reason);
                });
            }
        };

        try {
            // 立即执行 executor 函数
            executor(resolve, reject);
        } catch (error) {
            // 如果 executor 抛出异常，视为 Promise 被拒绝
            reject(error);
        }
    }

    then(onFulfilled, onRejected) {
        // 确保 onFulfilled 和 onRejected 是函数
        onFulfilled = typeof onFulfilled === 'function' ? onFulfilled :
value => value;
        onRejected = typeof onRejected === 'function' ? onRejected :
reason => { throw reason };

        // then 方法必须返回一个新的 Promise
        return new SimplePromise((resolve, reject) => {
            // 定义处理成功回调并处理其返回值的逻辑
            const handleFulfilled = (value) => {
                try {
                    // 执行成功回调
```

```
                    const result = onFulfilled(value);
                    // 根据回调的返回值决定新 Promise 的状态
                    // 这里的 Promise 解析过程（Promise Resolution
Procedure）是 Promise/A+ 标准的关键
                    // 简化版本只处理返回普通值，不处理返回 Promise 的情况
                     resolve(result);
                } catch (error) {
                    reject(error);
                }
            };

            // 定义处理失败回调并处理其返回值的逻辑
            const handleRejected = (reason) => {
                 try {
                    // 执行失败回调
                    const result = onRejected(reason);
                    // 失败回调返回普通值时，新 Promise 状态变为 fulfilled
                    resolve(result);
                } catch (error) {
                    // 失败回调抛出异常时，新 Promise 状态变为 rejected
                    reject(error);
                }
            };


            if (this.state === 'fulfilled') {
                // 如果当前 Promise 已成功，立即执行成功回调
                // 使用 setTimeout 模拟异步，符合 Promise/A+ 的微任务要求（简化
版用 setTimeout 即可）
                 setTimeout(() => {
                    handleFulfilled(this.value);
                 }, 0);
            } else if (this.state === 'rejected') {
                // 如果当前 Promise 已失败，立即执行失败回调
                 setTimeout(() => {
                    handleRejected(this.reason);
                 }, 0);
            } else { // this.state === 'pending'
                // 如果当前 Promise 仍在等待，将回调添加到队列
                this.onFulfilledCallbacks.push(() => {
                    setTimeout(() => {
                        handleFulfilled(this.value);
                     }, 0);
                });
                this.onRejectedCallbacks.push(() => {
                     setTimeout(() => {
                        handleRejected(this.reason);
                     }, 0);
                });
            }
```

```
        });
    }
}

// 使用示例:
new SimplePromise((resolve, reject) => {
    setTimeout(() => {
        const num = Math.random();
        if (num > 0.5) {
            resolve("成功: " + num);
        } else {
            reject("失败: " + num);
        }
    }, 1000);
})
.then(result => {
    console.log("第一个then - 成功:", result);
    return result + " -> 链式调用"; // 返回一个普通值
}, error => {
     console.log("第一个then - 失败:", error);
     throw new Error("新的错误: " + error); // 抛出错误
})
.then(result => {
    console.log("第二个then - 成功:", result);
})
.catch(error => { // 简化版可能不实现 catch，但概念上它是 .then(undefined, onRejected) 的语法糖
     console.log("全局catch - 捕获到错误:", error);
});
```

注意：以上代码是一个高度简化的版本，省略了 Promise/A+ 标准中的很多复杂细节，如 Promise Resolution Procedure (x 的解析过程，处理 thenable 和自身循环引用等)，以及 `catch`，`finally`，static methods (like `all`，`race`) 的实现。但它展示了状态管理、回调注册和基本链式调用的核心逻辑。

4. 关键注意事项 (Key Considerations)

1. **状态不可逆:** 一旦 Promise 的状态从 `'pending'` 变为 `'fulfilled'` 或 `'rejected'`，它就不能再次改变。
2. **异步执行回调:** 即使 Promise 在调用 `then` 时已经确定了状态，其回调函数也必须通过异步方式（微任务或宏任务，例如 `setTimeout(..., 0)`）执行，以确保 `then` 方法本身是同步返回 Promise 实例的。这符合 Promise/A+ 标准中关于时序的要求。
3. **then 方法的返回值:** `then` 方法**必须**返回一个新的 Promise 实例。这个新的 Promise 的状态和值/原因由回调函数的返回值决定。这是实现 Promise 链式调用的基础。
4. **Executor 函数的立即执行:** 传递给 Promise 构造函数的 `executor` 函数会立即同步执行。如果在其中抛出异常，Promise 会立即被拒绝。

5. 参考资料 (References)

- **MDN Web Docs - Promise:** https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Promise (官方文档)
- **Promises/A+ 规范:** https://promisesaplus.com/ (Promise 的权威技术规范，手写实现的重要参考)
- **JavaScript Promise 迷你书（中文版）:** https://liubin.org/promises-book/ (业界广泛参考的 Promise 学习资料)
- **深入理解 ECMAScript 6 - Promise:** https://es6.ruanyifeng.com/#docs/promise (阮一峰老师的ES6教程，对Promise有详细介绍)
