const PENDING = 'pending';

笔记本:	1.有关印象笔记		
创建时间:	2020/7/19 23:36	更新时间:	2020/7/19 23:36
作者:	1639079350@qq.com		
标签:	11.源码		

```
const PENDING = 'pending';
const FULFILLED = 'fulfilled';
const REJECTED = 'rejected';

class myPromise {
    constructor(executor) {
        try {
            executor(this.resolve, this.reject);

        } catch (e) {
            this.reject(e);
        }
    }
    status = PENDING; // 当前状态
    value = undefined; // 成功的值
    reason = undefined;
    successCallback = []; // 成功回调
    failCallback = [];
    resolve = value => {
        let promise2 = new myPromise(() => {
            // 如果状态不是等待，就不向下执行
            if (this.status !== PENDING) return false;
            this.status = FULFILLED;
            // 保存成功回调的值给then
            this.value = value;
            // 判断成功回调是否存在，存在即调用
            // this.successCallback && this.successCallback(this.value);
            while (this.successCallback.length) { // 异步
                this.successCallback.shift()(); // 依次弹出执行每个成功函数
            }
        })
        return promise2;
    }
    reject = reason => {
        if (this.status !== PENDING) return false;
        this.status = REJECTED;
        this.reason = reason;
        while (this.failCallback.length) { // 异步
            this.failCallback.shift()(); // 依次弹出执行每个成功函数
        }
    }
    then(successCallback, failCallback) {
        successCallback = successCallback ? successCallback : value => value;
```

```
        failCallback = failCallback ? failCallback : reason => { throw reason };
        let promise2 = new myPromise((resolve, reject) => {
            if (this.status === FULFILLED) {
                setTimeout(() => {
                    try {
                        let x = successCallback(this.value);
                        resolvePromise(promise2, x, resolve, reject);
                    } catch (e) {
                        reject(e);
                    }
                }, 0)
            } else if (this.status === REJECTED) {
                setTimeout(() => {
                    try {
                        let x = failCallback(this.reason);
                        resolvePromise(promise2, x, resolve, reject);
                    } catch (e) {
                        reject(e);
                    }
                }, 0)
            } else { // 等待
                this.successCallback.push(() => {
                    setTimeout(() => {
                        try {
                            let x = successCallback(this.value);
                            resolvePromise(promise2, x, resolve, reject);
                        } catch (e) {
                            reject(e);
                        }
                    }, 0)

                });
                this.failCallback.push(() => {
                    setTimeout(() => {
                        try {
                            let x = failCallback(this.reason);
                            resolvePromise(promise2, x, resolve, reject);
                        } catch (e) {
                            reject(e);
                        }
                    }, 0)

                });
            }
        })
        return promise2;
    }
    finally(callback) {
        // 对返回值进行统一处理，可处理异步问题
        // 如果是普通值，转化为promise对象，等待promise对象完成
        // 如果是promise对象，等待promise对象完成
        return this.then( value => {
            return myPromise.realove(callback().then(() => value));
```

```
            }, reason => {
                 return myPromise.realove(callback().then(() => { throw reason}));
            })
        }
        catch(failback) {
            return this.then(undefined,failback);
        }
        static all(array) {
            let result = [];
            let index = 0; // 防止异步，用来和执行的数组参数做比较
            return new MyPromise((resolve, reject) => {
                function addData(key, value) {
                    result[key] = value;
                    index++;
                    if (index === array.length) {
                        resolve(result);
                    }
                }
            })
            for (let i = 0; i < array.length; i++) {
                let current = array[i];
                if (current instanceof MyPromise) {
                    // promise对象
                    current.then(value => addData(i, value), reason => reject(reason));
                } else {
                    addData(i, array[i]);
                }
            }
        }
        static resolve(value) {
            if(value instanceof myPromise) return value;
            return new myPromise(resolve => resolve(value));
        }
    }
```