1. util.promisify方法进行promise风格转换

// 引入 util 模块

const util = require('util');

// 引入 fs 模块

const fs = require('fs');

// 返回一个新的函数

let mineReadFile = util.promisify(fs.readFile);

mineReadFile('./4resource/content.txt').then(value => {

    console.log(value.toString());

})

1. 状态和对象的值

**## Promise 的状态**

实例对象中的一个属性 『PromiseState』

\* pending  未决定的

\* resolved / fullfilled  成功

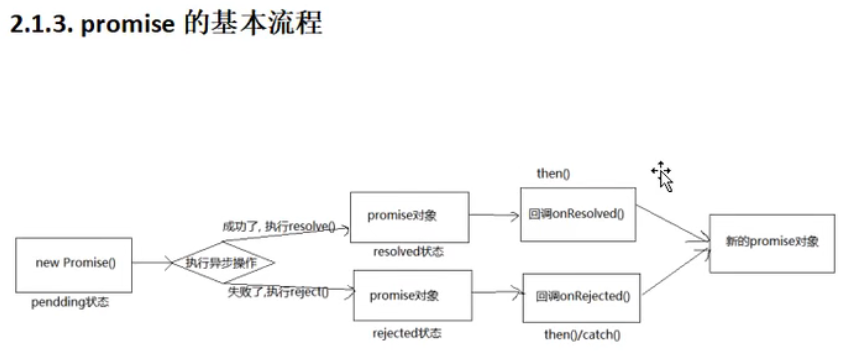
\* rejected 失败

**## Promise 对象的值**

实例对象中的另一个属性 『PromiseResult』

保存着异步任务『成功/失败』的结果

1. 执行过程



1. 同步调用

        let p = new Promise((resolve, reject) => {

            // \*\* 同步调用的

            reject('error');

        })

1. 中断

p.then(value => {

            return new Promise(() => {})

        })

1. 自定义Promise

function Promise(executor) {

    this.PromiseState = 'pending';

    this.PromiseResult = null;

    this.callbacks = [];

    const self = this;

    function resolve(data) {

        if(self.PromiseState !== 'pending') return;

        self.PromiseState = 'fulfilled';

        self.PromiseResult = data;

        setTimeout(() => {

            self.callbacks.forEach(item => {

                item.onResolved(data);

            })

        })

    }

    function reject(data) {

        if(self.PromiseState !== 'pending') return;

        self.PromiseState = 'rejected';

        self.PromiseResult = data;

        setTimeout(() => {

            self.callbacks.forEach(item => {

                item.onRejected(data);

            })

        })

    }

    try{

        executor(resolve, reject);

    } catch(e) {

        reject(e);

    }

}

Promise.prototype.then = function(onResolved, onRejected) {

    const self = this;

    if(typeof onRejected !== 'function') {

        onRejected = reason => {

            throw reason;

        }

    }

    if(typeof onResolved !== 'function') {

        onResolved = value => value;

    }

    return new Promise((resolve, reject) => {

        function callback(type) {

            try{

                let result = type(self.PromiseResult);

                if (result instanceof Promise) {

                    result.then(v => {

                        resolve(v);

                    }, r => {

                        reject(r);

                    })

                } else {

                    resolve(result);

                }

            }catch(e) {

                reject(e);

            }

        }

        if(this.PromiseState === 'fulfilled'){

            setTimeout(() => {

                callback(onResolved);

            })

        }

        if(this.PromiseState === 'rejected') {

            setTimeout(() => {

                callback(onRejected);

            })

        }

        if(this.PromiseState === 'pending') {

            this.callbacks.push({

                onResolved: function(value) {

                    callback(onResolved);

                },

                onRejected: function(value) {

                    callback(onRejected)

                }

            })

        }

    })

}

Promise.prototype.catch = function(onRejected) {

    return this.then(undefined, onRejected);

}

Promise.resolve = function(value) {

    return new Promise((resolve, reject) => {

        if (value instanceof Promise) {

            value.then(v => {

                resolve(v);

            }, r => {

                reject(r);

            })

        } else {

            resolve(value);

        }

    })

}

Promise.reject = function(reason) {

    return new Promise((resolve, reject) => {

        reject(reason);

    })

}

Promise.all = function(promises) {

    return new Promise((resolve, reject) => {

        let count = 0;

        let arr = [];

        for(let i =0; i < promises.length; i++) {

            promises[i].then(v => {

                count++;

                arr[i] = v;

                if(count === promises.length) {

                    resolve(arr);

                }

            }, r => {

                reject(r);

            })

        }

    })

}

Promise.race = function(promises) {

    return new Promise((resolve, reject) => {

        for(let i = 0; i < promises.length; i++) {

            promises[i].then(v => {

                resolve(v)

            }, r => {

                reject(r);

            })

        }

    })

}

1. async与await发送AJAX

function sendAJAX(url) {

            return new Promise((resolve, reject) => {

                // ...

            })

        }

        let btn = document.querySelector('#btn');

        btn.addEventListener('click', async function() {

            let duanzi = await sendAJAX('https://api.apiopen.top/getJoke');

            console.log(duanzi);

        })