function Promise(fn) {

var state = 'pending', // Promise状态

value = null, // 当前reolve传入的原始变量值

callbacks = []; // 回调方法队列

// then方法返回一个新的Promise, 新的Promise中调用一次 handle方法, 要注意handle中传入的固定参数格式

this.then = function (onFulfilled, onRejected) {

return new Promise(function (resolve, reject) {

handle({

onFulfilled: onFulfilled || null,

onRejected: onRejected || null,

resolve: resolve,

reject: reject

});

});

};

function handle(callback) {

if (state === 'pending') { // 如果当前Promise的状态是pedding

callbacks.push(callback); // 将callback函数添加到callbacks回调队列中

return; // 直接返回

}

// 如果状态是不是 pedding，也就是状态发生了改变

// 如果当前状态是 fulfilled, cd == callback.onFulfilled

// 如果当前状态是 rejected, cd == callback.onRejected

var cb = state === 'fulfilled' ? callback.onFulfilled : callback.onRejected,

ret;

// 如果then中传入的参数 onFulfilled(onRejected) 为假, 或者说没有传入参数, 那么 cd == callback.resolve(reject), 也就是then中返回的Promise中传入的fn的参数resolve,然后resolve(value),然后直接返回

if (cb === null) {

cb = state === 'fulfilled' ? callback.resolve : callback.reject;

cb(value);

return;

}

// 如果callback.onFulfilled 或 callback.onRejected不为空且不是一个函数(promise/A+规范规定,onFulfilled(onRejected)不是一个函数必须被忽略)

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

state === 'fulfilled' ? callback.resolve(value) : callback.reject(value);

return;

}

// 当callback.onFulfilled 或 callback.onRejected是一个函数, 执行报错时, 调用callback.reject 将状态更改为reject，将错误作为拒因传递

try {

ret = cb(value);

callback.resolve(ret);

} catch (e) {

callback.reject(e);

}

}

function resolve(newValue) {

// 如果newValue是一个对象或方法

if (newValue && (typeof newValue === 'object' || typeof newValue === 'function')) {

var then = newValue.then; // 获取 thenable 的then

// 如果 then 是一方法, 也就是 newValue是一个thenable, then递归调用 resolve、reject方法,然后直接返回

if (typeof then === 'function') {

// 一般来说thenable的then方法,都需要传入一个或两个方法(假装自己是一个Promisede的真正的then方法),所以then.call(newValue,resolve,reject)中的resolve,reject就是传入的两个方法,用来调用resolve或reject,随后改变状态

then.call(newValue, resolve, reject);

return;

}

}

state = 'fulfilled'; // 调整当前Promise的状态

value = newValue; // 如果newValue不是一个thenable，设置当前value的值为newValue

execute(); // 延迟调用回调队列中的函数

}

function reject(reason) {

state = 'rejected'; // 调整当前Promise的状态

value = reason; // 设置当前value的值为reason

execute(); // 延迟调用回调队列中的函数

}

// 延迟调用函数，避免同步立即执行callbacks回调队列中的函数

function execute() {

setTimeout(function () {

callbacks.forEach(function (callback) {

handle(callback);

});

}, 0);

}

fn(resolve, reject);

}

// 测试

var a = new Promise( function(resolve, reject){

console.log(1);

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

reject('error');

}));

} ).then(function(data){

console.log(data);

return new Promise(function(resolve,reject){

resolve(4);

});

}, function(err){

console.log(err);

return new Promise(function(resolve,reject){

resolve(4);

});

}).then(data => {

console.log(data);

a.a();

}).then(1)

.then( data=>{

console.log(data);

}, err=>{

console.log(err);

} );