Promise构造函数

**一.Promise是一个构造函数，需要new**

**new完以后不调用也执行，所以一般会将其封装在函数里**

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

resolve('成功');

reject('失败');

**})**

**二.Promise对象有以下2个特点**

**1.**对象的状态不受外界影响。Promise对象代表一个异步操作，有三种状态：Pending(进行中)、Resolved(已完成)和Rejected(已失败)。只有异步操作的结果，可以决定当前是哪一种状态，任何其他操作都无法改变这个状态。这也是Promise这个名字的由来，它的英语意思就是“承诺”，表示其他手段无法改变。

**2.**一旦状态改变，就不会再变，任何时候都可以得到这个结果。Promise对象的状态改变，只有两种可能：从Pending变为Resolved；从Pending变为Rejected。只要这两种情况发生，状态就凝固了，不会再变了，会一直保持这个结果。就算改变已经发生了，你再对Promise对象回调函数，也会立即得到这个结果。这与事件(Event)完全不同，事件的特点是，如果你错过了它，再去监听，是得不到结果的。

**三.then**

.then(A,B)有AB两个**函数**作为参数，A接受resolve的数据,B接受reject的数据

**四.catch**

then里面不执行，就执行catch里面的

**五.all**

Promise.all来执行，all接收一个**数组参数**，这组参数为需要执行异步操作的所有方法，里面的值最终都算返回Promise对象。这样，**多个异步操作的并行执行的**，等到它们都执行完后才会进到**then里面**。all会把所有异步操作的结果放进一个数组中传给then，然后再执行then方法的成功回调将结果接收

**六.race**

race方法**谁先执行完成就先执行回调**。先执行完的不管是进行了race的成功回调还是失败回调，其余的将不会再进入race的任何回调

**七.promise用途**

**1**.用于异步计算；

**2**.可以将异步操作队列化，按照期望的顺序执行，返回符合预期的结果；

**3**.可以在对象之间传递和操作promise，帮助我们处理队列。

function runAsync1() {

console.log('开始')

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

setTimeout(function () {

console.log('异步任务1执行完成');

resolve('获得数据1成功');

}, 0)

});

return p;

}

function runAsync2() {

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

setTimeout(function () {

console.log('异步任务2执行完成');

resolve('获得数据2成功');

}, 0);

});

return p;

}

function runAsync3() {

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

setTimeout(function () {

console.log('异步任务3执行完成');

resolve('获得数据3成功');

}, 0);

});

return p;

}

**race的用法：**

function startRace() {

Promise.race([runAsync1(), runAsync2(), runAsync3()]).**then(**function (data) {

console.log(data);//只打印1的参数

})

}

**all的用法：**

function startAll() {

Promise.all([runAsync1(), runAsync2(), runAsync3()]).**then**(function (data) {

console.log(data[0]);

console.log(data[1]);

console.log(data[2]);

})

}

**then的用法：**

function startThen() {

runAsync1().then( //第一步

function (data) {

console.log(data);

return runAsync2();

}

).then(function (data) { //第二步

console.log(data);

return runAsync3();

}).then(function (data) { //第三步

console.log(data);

})

}

function getNum() {

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

setTimeout(function () {

var num = Math.ceil(Math.random() \* 10);

if (num < 5) {

console.log('获取成功');

resolve(num);

} else {

console.log('获取失败');

reject(['数值太大', num]);

}

}, 10)

});

return p;

}

**catch的用法：**

function startCatch() {

getNum().then( //then(reslve,reject)

function (num) {

console.log('resolved')

console.log(num)

},

function ([reason, num]) {

console.log('rejected');

console.log(reason);

console.log(num);

}

).catch(function (data) {

console.log(data) //上面不执行来到这里，等于then的第二个参数

})

}

**请求图片资源成功**

function requestImg(url) {

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

setTimeout(function () {

var img = new Image(300);

img.onload = function () {

resolve(img);

var div = document.getElementById('aaa');

div.appendChild(img)

}

img.src = url;

}, 2000)

});

return p;

}

**加载不到这个图片**

function timeOut() {

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

setTimeout(function () {

reject('请求图片失败');

}, 5000)

});

return p;

}

Promise.race([requestImg('subway.jpg'), timeOut()]).then(function (img) {

console.log(img)

}).catch(function (reason) {

console.log(reason)

})

**封装MySimplePromise**

**1.三大块：this.then,resolve/reject,fn(resolve,reject)**

**2.this.then负责注册函数所有的函数，reslove/reject负责执行所有的函数**

**3.在resolve/reject里要加上setTimeout防止还没有进行then注册，就直接执行resolve**

**4resolve/reject里面返回this，这样就可以链式调用了**

**5.三个状态的管理（pending，fulfilled，rejected）**

**6.promise的链式调用在then里面return一个promise这样才可以then里面加上异步函数**

**7.加上catch**

function MySimplePromise(fn) {

var value = null;

var callBacks = [ ];//回调函数

//加入了状态，为了解决在promise异步操作成功后调用的then注册的回调不会执行的问题

var state = 'pending';//状态

var \_this = this;//指向函数体

//注册所有回调函数

this.then = function (fulfilled, rejected) {

//如果想要链式promise那就要在这边return一个new Promise

return new MySimplePromise(function (resolve, reject) {

//异常处理

try {

if (state == 'pending') {

callBacks.push(fulfilled);

return; //实现链式调用

}

if (state == 'fulfilled') {

var data = fulfilled(value);

//为了让两个promise连接起来

resolve(data);

return

}

if (stated == 'rejected') {

var data = rejected(value);

//为了让两个promise连接起来

resolve(data);

return;

}

} catch (e) {

\_this.catch(e)

}

});

}

//执行所有回调函数

function resolve(valueNew){

value= valueNew;

state = 'fulfilled';

execute();

}

//执行所有回调函数

function reject(valueNew){

value= valueNew;

state = 'rejected';

execute();

}

//在resolve/reject里要加上setTimeout防止还没有进行then注册，就直接执行resolve

function execute(){

setTimeout(function(){

callBacks.forEach(function(cb){

value = cb(value)

})

},0);

}

this.catch = function(e){

console.log(JSON.stringify(e));

}

//实现异步回调

fn(resolve,reject);

}