Promise

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;

}

function startRace() {

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

console.log(data);

})

}

function startAll() {

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

console.log(data[0]);

console.log(data[1]);

console.log(data[2]);

})

}

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;

}

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);

}