promise的使用

1. 实例化promise对象

// 立即执行

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

resolve(1);

})

// 延迟执行

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

setTimeout(function(){

var n = 2;

resolve(n);

}, 1000)

})

// 以延迟执行为例

// then一次运行，延迟1秒执行，console为2

promise.then(

function(n){console.log(n)}

)

1. promise的多次执行

// promise定义

var promise1 = new Promise(function(resolve, reject){

setTimeout(function(){

var n = 2;

resolve(n);

}, 1000)

})

var promise2 = new Promise(function(resolve, reject){

setTimeout(function(){

var n = 2;

resolve(n);

}, 1000)

})

// 第一次延迟1秒执行，返回1，

// 第二次跟随其后立即执行，返回undefined

promise1.then(

function(n1){console.log(n1)}

).then(

function(n2){console.log(n2)}

)

// 第一次延迟1秒执行，返回1，

// 第二次跟随其后立即执行，返回2

// 原因：promise1和promise2在定义时，构造函数已经执行了当中的函数

// 当中的定时器已经开始执行，两次promise的延迟均为1秒，

// 故promise1完成时，promise2也已经完成会立即执行

promise1.then(

function(n1){

console.log(n1);

return promise2;

}

).then(

function(n2){console.log(n2)}

)

// 第一次延迟1秒执行，返回1，

// 第二次延迟1秒执行，返回3

// 原因：promise3在promise1执行完成后定义，所以会依次延迟执行  
promise1.then(

function(n1){

console.log(n1);  
 var promise3 = new Promise(function(resolve, reject){

setTimeout(function(){

var n = 3;

resolve(n);

}, 1000)

})

return promise3;

}

).then(

function(n2){console.log(n2)}

)