# #### 1、深拷贝浅拷贝

浅拷贝：浅拷贝通过ES6新特性Object.assign()或者通过扩展运算法...来达到浅拷贝的目的，浅拷贝修改副本，不会影响原数据，但缺点是浅拷贝只能拷贝第一层的数据，且都是值类型数据，如果有引用型数据，修改副本会影响原数据。

深拷贝：通过利用JSON.parse(JSON.stringify())来实现深拷贝的目的，但利用JSON拷贝也是有缺点的，

当要拷贝的数据中含有undefined/function/symbol类型是无法进行拷贝的，当然我们想项目开发中需要深拷贝的数据一般不会含有以上三种类型，如有需要可以自己在封装一个函数来实现。

# #### 1 什么是解构赋值

解构赋值允许你使用类似数组或对象字面量的语法将数组和对象的属性赋给各种变量。这种赋值语法极度简洁，同时还比传统的属性访问方法更为清晰。解构会仅提取出一个大对象中的部分成员来单独使用

# #### 2、ES6新特性

1.ES6引入来严格模式

变量必须声明后在使用

函数的参数不能有同名属性, 否则报错

不能使用with语句 (说实话我基本没用过)

不能对只读属性赋值, 否则报错

不能使用前缀0表示八进制数,否则报错 (说实话我基本没用过)

不能删除不可删除的数据, 否则报错

不能删除变量delete prop, 会报错, 只能删除属性delete global[prop]

eval不会在它的外层作用域引入变量

eval和arguments不能被重新赋值

arguments不会自动反映函数参数的变化

不能使用arguments.caller (说实话我基本没用过)

不能使用arguments.callee (说实话我基本没用过)

禁止this指向全局对象

不能使用fn.caller和fn.arguments获取函数调用的堆栈 (说实话我基本没用过)

增加了保留字（比如protected、static和interface）

2.关于let和const新增的变量声明

3.变量的解构赋值

4.字符串的扩展

includes()：返回布尔值，表示是否找到了参数字符串。

startsWith()：返回布尔值，表示参数字符串是否在原字符串的头部。

endsWith()：返回布尔值，表示参数字符串是否在原字符串的尾部。

5.数值的扩展

Number.isFinite()用来检查一个数值是否为有限的（finite）。

Number.isNaN()用来检查一个值是否为NaN。

6.函数的扩展

函数参数指定默认值

7.数组的扩展

扩展运算符

8.对象的扩展

对象的解构

9.新增symbol数据类型

10.Set 和 Map 数据结构

ES6 提供了新的数据结构 Set。它类似于数组，但是成员的值都是唯一的，没有重复的值。 Set 本身是一个构造函数，用来生成 Set 数据结构。

Map它类似于对象，也是键值对的集合，但是“键”的范围不限于字符串，各种类型的值（包括对象）都可以当作键。

11.Proxy

Proxy 可以理解成，在目标对象之前架设一层“拦截”，外界对该对象的访问

都必须先通过这层拦截，因此提供了一种机制，可以对外界的访问进行过滤和改写。

Proxy 这个词的原意是代理，用在这里表示由它来“代理”某些操作，可以译为“代理器”。

Vue3.0使用了proxy

12.Promise

Promise 是异步编程的一种解决方案，比传统的解决方案——回调函数和事件——更合理和更强大。

特点是：

对象的状态不受外界影响。

一旦状态改变，就不会再变，任何时候都可以得到这个结果。

13.async 函数

async函数对 Generator 函数的区别：

（1）内置执行器。

Generator 函数的执行必须靠执行器，而async函数自带执行器。也就是说，async函数的执行，与普通函数一模一样，只要一行。

（2）更好的语义。

async和await，比起星号和yield，语义更清楚了。async表示函数里有异步操作，await表示紧跟在后面的表达式需要等待结果。

（3）正常情况下，await命令后面是一个 Promise 对象。如果不是，会被转成一个立即resolve的 Promise 对象。

（4）返回值是 Promise。

async函数的返回值是 Promise 对象，这比 Generator 函数的返回值是 Iterator 对象方便多了。你可以用then方法指定下一步的操作。

14.Class

class跟let、const一样：不存在变量提升、不能重复声明...

ES6 的class可以看作只是一个语法糖，它的绝大部分功能

ES5 都可以做到，新的class写法只是让对象原型的写法更加清晰、更像面向对象编程的语法而已。

15.Module

ES6 的模块自动采用严格模式，不管你有没有在模块头部加上"use strict";。

import和export命令以及export和export default的区别

# #### 3、es5和es6的区别，说一下你所知道的es6

　　ECMAScript5，即ES5，是ECMAScript的第五次修订，于2009年完成标准化

　　ECMAScript6，即ES6，是ECMAScript的第六次修订，于2015年完成，也称ES2015

ES6是继ES5之后的一次改进，相对于ES5更加简洁，提高了开发效率

# #### 4、ES6新增的一些特性：

　　1）let声明变量和const声明常量，两个都有块级作用域  
　　ES5中是没有块级作用域的，并且var有变量提升，在let中，使用的变量一定要进行声明

　　2）箭头函数  
　　ES6中的函数定义不再使用关键字function()，而是利用了()=>来进行定义

　　3）模板字符串  
　　模板字符串是增强版的字符串，用反引号（`）标识，可以当作普通字符串使用，也可以用来定义多行字符串

　　4）解构赋值  
　　ES6 允许按照一定模式，从数组和对象中提取值，对变量进行赋值

　　5）for of循环  
　　for...of循环可以遍历数组、Set和Map结构、某些类似数组的对象、对象，以及字符串

　　6）import、export导入导出  
　　ES6标准中，Js原生支持模块(module)。将JS代码分割成不同功能的小块进行模块化，将不同功能的代码分别写在不同文件中，各模块只需导出公共接口部分，然后通过模块的导入的方式可以在其他地方使用

　　7）set数据结构  
　　Set数据结构，类似数组。所有的数据都是唯一的，没有重复的值。它本身是一个构造函数

　　8）... 展开运算符   
　　可以将数组或对象里面的值展开；还可以将多个值收集为一个变量

　　9）修饰器 @  
　　decorator是一个函数，用来修改类甚至于是方法的行为。修饰器本质就是编译时执行的函数

　　10）class 类的继承  
　　ES6中不再像ES5一样使用原型链实现继承，而是引入Class这个概念

　　11）async、await  
　　使用 async/await, 搭配promise,可以通过编写形似同步的代码来处理异步流程, 提高代码的简洁性和可读性  
　　async 用于申明一个 function 是异步的，而 await 用于等待一个异步方法执行完成

　　12）promise  
　　Promise是异步编程的一种解决方案，比传统的解决方案（回调函数和事件）更合理、强大

　　13）Symbol  
　　Symbol是一种基本类型。Symbol 通过调用symbol函数产生，它接收一个可选的名字参数，该函数返回的symbol是唯一的

　　14）Proxy代理  
　　使用代理（Proxy）监听对象的操作，然后可以做一些相应事情

# #### 5、var、let、const之间的区别

　　var声明变量可以重复声明，而let不可以重复声明

　　var是不受限于块级的，而let是受限于块级

　　var会与window相映射（会挂一个属性），而let不与window相映射

　　var可以在声明的上面访问变量，而let有暂存死区，在声明的上面访问变量会报错

　　const声明之后必须赋值，否则会报错

　　const定义不可变的量，改变了就会报错

　　const和let一样不会与window相映射、支持块级作用域、在声明的上面访问变量会报错

# #### 6、使用箭头函数应注意什么？

　　（1）用了箭头函数，this就不是指向window，而是父级（指向是可变的）

　　（2）不能够使用arguments对象

　　（3）不能用作构造函数，这就是说不能够使用new命令，否则会抛出一个错误

　　（4）不可以使用yield命令，因此箭头函数不能用作 Generator 函数

# #### 7、ES6的模板字符串有哪些新特性？并实现一个类模板字符串的功能

　　基本的字符串格式化。将表达式嵌入字符串中进行拼接。用${}来界定

　　在ES5时我们通过反斜杠()来做多行字符串或者字符串一行行拼接。ES6反引号(``)就能解决

类模板字符串的功能

let name = 'web';

let age = 10;

let str = '你好，${name} 已经 ${age}岁了'

str = str.replace(/\$\{([^}]\*)\}/g,function(){

return eval(arguments[1]);

})

console.log(str);//你好，web 已经 10岁了

# #### 8、介绍下 Set、Map的区别？

　　应用场景Set用于数据重组，Map用于数据储存

　　Set：

　　（1）成员不能重复  
　　（2）只有键值没有键名，类似数组  
　　（3）可以遍历，方法有add, delete,has

　　Map:

　　（1）本质上是健值对的集合，类似集合  
　　（2）可以遍历，可以跟各种数据格式转换

# #### 9、ES 6 怎么写 class ，为何会出现 class？

　　ES6的class可以看作是一个语法糖，它的绝大部分功能ES5都可以做到，新的class写法只是让对象原型的写法更加清晰、更像面向对象编程的语法

//定义类class Point {

constructor(x,y) {

//构造方法

this.x = x; //this关键字代表实例对象

this.y = y;

} toString() {

return '(' + this.x + ',' + this.y + ')';

}

}

# #### 10、说一下es6的导入导出模块

　　导入通过import关键字

// 只导入一个

import {sum} from "./example.js"

// 导入多个

import {sum,multiply,time} from "./exportExample.js"

// 导入一整个模块

import \* as example from "./exportExample.js"

[IMG_265](https://www.cnblogs.com/theblogs/p/javascript:void(0);)

　　导出通过export关键字

[IMG_266](https://www.cnblogs.com/theblogs/p/javascript:void(0);)

//可以将export放在任何变量,函数或类声明的前面

export var firstName = 'Michael';

export var lastName = 'Jackson';

export var year = 1958;

//也可以使用大括号指定所要输出的一组变量var firstName = 'Michael';var lastName = 'Jackson';var year = 1958;

export {firstName, lastName, year};

//使用export default时，对应的import语句不需要使用大括号

let bosh = function crs(){}

export default bosh;

import crc from 'crc';

//不使用export default时，对应的import语句需要使用大括号

let bosh = function crs(){}

export bosh;

import {crc} from 'crc';

# #### 11、[ES6常用语法总结](https://www.cnblogs.com/zhoulifeng/p/10688647.html)

ECMAScript 6（以下简称ES6）是JavaScript语言的下一代标准。因为当前版本的ES6是在2015年发布的，所以又称ECMAScript 2015。也就是说，ES6就是ES2015。虽然目前并不是所有浏览器都能兼容ES6全部特性，但越来越多的程序员在实际项目当中已经开始使用ES6了。今天就来总结一下在工作中es6常用的属性方法

1.let

ES6新增了let命令，用来生命变量。它的用法类似于var,但是所声明的变量，只在let命令所在的代码块内有效。

|  |  |
| --- | --- |
| 1  2  3  4 | for (let i = 0; i < 10; i++) {     }      console.log(i) //ReferenceError: i is not defined<br><br>for(var i=0;i<10;i++){<br><br>}<br>console.log(i) //10<br> |

　上面代码中，计数器i只在for循环体内有效，在循环体外引用就会报错。let具有块级作用域的。var不存在块级作用域问题，具有全局变量提示的问题存在，

2.const

const声明一个只读的常量，一旦声明，常量的值就不能改变。

const a = 10；

a = 20；

console.log(a) //TypeError: Assignment to constant variable.

上面代码表明改变常量的值会报错。

const声明的变量不得改变值，这意味着，const一旦声明变量，就必须立即初始化，不能留到以后赋值。

const a;

console.log(a)

上面代码表示，对于const来说，只声明不赋值，就会报错。

let与const

　1.都不能重复声明

　2.都存在块级作用域问题

   3.只在声明所在的块级作用域内有效。

3.模板字符串

模板字符串就是一种字符串的新的表现形式

（1）基本用法

let s1 = ` hello `

let s2 = ' hello '

（2）字符串和变量拼接

let s3 =" a " + s1 + " b " + s2;

let s4 = ` a ${s1} b ${s2}`;

(3)字符串换行

var box =`<div>

<p>

<span>123</span>

</p>

<p>${a1}</p>

</div>`;

模板字符串的出现，极大的改变传统字符串的拼接方法，减少代码出现错误的几率。提高开发效率

4.解构赋值

ES6 允许按照一定模式，从数组和对象中提取值，对变量进行赋值，这被称为解构

　　4.1:对象结构赋值

[IMG_256](https://www.cnblogs.com/zhoulifeng/p/javascript:void(0);)

var obj ={ name:"abc",age:18 };

//用解构赋值的方式获取name、age

let { name } = obj; //创建了一个变量name，值=obj.name

console.log(name); //"abc"

let { age } =obj;

console.log(age); //18

　 4.1:函数参数结构赋值

function f1(obj){

console.log(obj.age);

console.log(obj.height)

}

//等价于

function f1({ age,height }){

console.log(age);

console.log(height)

}

f1({age:5,height:180})

5.rest参数

ES6 引入 rest 参数（形式为...变量名），用于获取函数的多余参数，这样就不需要使用arguments对象了。rest 参数搭配的变量是一个数组，该变量将多余的参数放入数组中。

function fn(){

//arguments是函数内部的一个局部变量，

//arguments.length表示函数的实参的个数

console.log(arguments.length);

for(var i =0 ; i<arguments.length; i++){

console.log(arguments[i]);

}

}

fn(1,3,5) //3

// fn("a","b","c","d","e") //5

es6箭头函数内部不能使用arguments，为了弥补这个问题，rest参数应孕而生

//...args就是rest参数

//-->产生了一个变量，这个变量是一个数组，数组里面包含了这个函数调用时传递的所有实参

function q(...args){

//验证args是不是数组？

console.log(args instanceof Array);//true

console.log(Object.prototype.toString.call(args));//"[object Array]"

console.log(Array.isArray(args)); //true es5中的新方法

console.log(args);

}

q(1,3,5);

q(2,3,4,5,6,7,8);

[IMG_263](https://www.cnblogs.com/zhoulifeng/p/javascript:void(0);)

6.箭头函数

ES6 允许使用“箭头”（=>）定义函数。

场景：用于替换匿名函数

基本用法  
//匿名函数

div.onclick=function(){

console.log("你好")

}

//箭头函数

div.onclick=()=>{

console.log("你好")

}

[IMG_265](https://www.cnblogs.com/zhoulifeng/p/javascript:void(0);)

有一个参数的箭头函数

[IMG_266](https://www.cnblogs.com/zhoulifeng/p/javascript:void(0);)

var fn=(a)=>{

console.log("abc");

}

//等价于：

var fn=a=>{

console.log("abc");

}

[IMG_267](https://www.cnblogs.com/zhoulifeng/p/javascript:void(0);)

有2个及更多参数的箭头函数

var f=(a,b,c)=>{

console.log("abc")

}

[IMG_268](https://www.cnblogs.com/zhoulifeng/p/javascript:void(0);)

var p={

age:18,

//es6中对象方法的箭头函数表示形式

run:()=>{

setTimeout(()=>{

//this:window

console.log(this);//this是window

},100)

},

travel:function(){

//this:p

setTimeout(()=>{

console.log(this);//this是p

},100)

},

//推荐使用的方式☆☆☆：es6中对象方法的简写形式

say(){

console.log("say方法中的this：",this);

setTimeout(()=>{

console.log("say内部的延迟函数：",this);//this是p

},100)

},

}

p.run();

p.travel();

p.say();

箭头函数和普通匿名函数有哪些不同？

　　1.函数体内的this对象，就是定义所在的对象，而不是使用所在的对象

　　2.不可以当作构造函数，也就是说，不可以使用new命令，否则会抛出一个错误

　　3.不可以使用arguments对象，该对象在函数体内不存在。如果要用，可以用 rest 参数代替。

　　4.不可以使用yield命令，因此箭头函数不能用作 Generator 函数。

 　5.generator函数现在经常用async替代

7.对象的扩展

扩展运算符（spread）是三个点（...）。用于取出参数对象的所有可遍历属性，拷贝到当前对象中

　　7.1Object.assign：实现拷贝继承

[IMG_270](https://www.cnblogs.com/zhoulifeng/p/javascript:void(0);)

//Object.assign 就是进行对象的浅拷贝

var source={ age:18,height:170,className:"3年2班" }

//克隆一个新对象出来

var newObj=Object.assign({},source);

console.log(newObj);

var newObj2={};

Object.assign(newObj2,source);

console.log(newObj2);

[IMG_271](https://www.cnblogs.com/zhoulifeng/p/javascript:void(0);)

上面可以实现浅拷贝，但是代码有点多，es6这个对象扩展，牛掰的一个方法，解决浅拷贝的问题，

[IMG_272](https://www.cnblogs.com/zhoulifeng/p/javascript:void(0);)

var car={ brand:"BMW",price:"368000",length:"3米" }

//克隆一个跟car完全一样的对象出来：

var car2={ ...car }

console.log(car2);

//新车子，跟car的长度不同，其他相同

var car3={ ...car,length:"4米" }

console.log(car3);

var car4={ ...car,type:"SUV"}

console.log(car4);

var car5={...car4,price:"69800",brand:"BYD"};

console.log(car5);

[IMG_273](https://www.cnblogs.com/zhoulifeng/p/javascript:void(0);)

对象扩展，简单方便，代码更加简介，更少的代码实现更强大的功能。

 8.Promise

Promise是异步编程一种解决方案(回调地狱)

[IMG_274](https://www.cnblogs.com/zhoulifeng/p/javascript:void(0);)

在没有promise都是这样写的回调，一层一层的写，  
$.get("/getUser",function(res){

$.get("/getUserDetail",function(){

$.get("/getCart",function(){

$.get("/getBooks",function(){

//...

})

})

})

})

[IMG_275](https://www.cnblogs.com/zhoulifeng/p/javascript:void(0);)

# #### 12、promise的基本用法

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

//b 把需要执行的异步操作放在这里

$.get("/getUser",res=>{

//获取数据的异步操作已经执行完毕了，等待下一步的执行，通过执行resolve函数，告诉外界你可以执行下一步操作了

//c、

resolve(res)

//而执行的下一步操作，其实就是写在then的回调函数中的

})

})

//a、

promise.then(res=>{

//d、执行后续的操作

console.log(res);

})

[IMG_277](https://www.cnblogs.com/zhoulifeng/p/javascript:void(0);)

promise实现多层回调

[IMG_278](https://www.cnblogs.com/zhoulifeng/p/javascript:void(0);)

new Promise((resolve,reject)=>{

$.get("/getUser",res=>{

resolve(res)

})

}).then(res=>{

//用户基本信息

return new Promise(resolve=>{

$.get("/getUserDetail",res=>{

resolve(res)

})

})

}).then(res=>{

//用户详情

return new Promise(resolve=>{

$.get("/getCart",res=>{

resolve(res)

})

})

}).then(res=>{

//购物车信息

})

[IMG_279](https://www.cnblogs.com/zhoulifeng/p/javascript:void(0);)

promise实现错误处理

[IMG_280](https://www.cnblogs.com/zhoulifeng/p/javascript:void(0);)

new Promise((resolve,reject)=>{

$.ajax({

url:"/getUser",

type:"GET",

success:res=>{

resolve(res);

},

error:res=>{

reject(res)

}

})

}).then(resSuccess=>{

//成功的返回值

},resError=>{

//失败的返回值

})

# #### 92、理解 async/await以及对Generator的优势

　　 async await 是用来解决异步的，async函数是Generator函数的语法糖

　　使用关键字async来表示，在函数内部使用 await 来表示异步

　　async函数返回一个 Promise 对象，可以使用then方法添加回调函数

　　当函数执行的时候，一旦遇到await就会先返回，等到异步操作完成，再接着执行函数体内后面的语句

　　async较Generator的优势：

　　（1）内置执行器。Generator 函数的执行必须依靠执行器，而 Aysnc 函数自带执行器，调用方式跟普通函数的调用一样

　　（2）更好的语义。async 和 await 相较于 \* 和 yield 更加语义化

　　（3）更广的适用性。yield命令后面只能是 Thunk 函数或 Promise对象，async函数的await后面可以是Promise也可以是原始类型的值

　　（4）返回值是 Promise。async 函数返回的是 Promise 对象，比Generator函数返回的Iterator对象方便，可以直接使用 then() 方法进行调用

# **#### 13、**promise和async/await区别

**区别：**

1 promise是ES6，async/await是ES7

2 async/await相对于promise来讲，写法更加优雅

3 reject状态：

1）promise错误可以通过catch来捕捉，建议尾部捕获错误，

2）async/await既可以用.then又可以用try-catch捕捉

promise和async/await都是处理异步请求

基本语法：

promise共有三个状态

链式调用

错误捕获

async、await

用法

错误捕获

区别：

promise的用法

Promise,简单来说就是一个容器，里面保存着某个未来才会结束的时间(通常是一个异步操作的结果)

基本语法：

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

//...

resolve('success')

});

p.then(result => {

console.log(result);//success

});

promise共有三个状态 pending（执行中）、success（成功）、rejected（失败）

链式调用

错误捕获

Promise.prototype.catch用于指定Promise状态变为rejected时的回调函数，可以认为是.then的简写形势，返回值跟.then一样

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

reject('error');

});

p.catch(result => {

console.log(result);

})`

async、await

简洁：异步编程的最高境界就是不关心它是否是异步。async、await很好的解决了这一点，将异步强行转换为同步处理。

async/await与promise不存在谁代替谁的说法，因为async/await是寄生于Promise，Generater的语法糖。

用法

async用于申明一个function是异步的，而await可以认为是async wait的简写，等待一个异步方法执行完成。

规则：

1 async和await是配对使用的，await存在于async的内部。否则会报错

2 await表示在这里等待一个promise返回，再接下来执行

3 await后面跟着的应该是一个promise对象，（也可以不是，如果不是接下来也没什么意义了…）

写法：

`async function demo() {

let result01 = await sleep(100);

//上一个await执行之后才会执行下一句

let result02 = await sleep(result01 + 100);

let result03 = await sleep(result02 + 100);

// console.log(result03);

return result03;

}

demo().then(result => {

console.log(result);

});`

错误捕获

如果是reject状态，可以用try-catch捕捉

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

setTimeout(() => {

reject('error');

},1000);

});

async function demo(params) {

try {

let result = await p;

}catch(e) {

console.log(e);

}

}

demo();

# #### 14、Promise 出现的原因

在 Promise 出现以前，我们处理一个异步网络请求，大概是这样：

// 请求 代表 一个异步网络调用。// 请求结果 代表网络请求的响应。

请求1(function(请求结果1){

处理请求结果1

})复制代码

看起来还不错。  
但是，需求变化了，我们需要根据第一个网络请求的结果，再去执行第二个网络请求，代码大概如下：

请求1(function(请求结果1){

请求2(function(请求结果2){

处理请求结果2

})

})复制代码

看起来也不复杂。  
但是。。需求是永无止境的，于是乎出现了如下的代码：

请求1(function(请求结果1){

请求2(function(请求结果2){

请求3(function(请求结果3){

请求4(function(请求结果4){

请求5(function(请求结果5){

请求6(function(请求结果3){

...

})

})

})

})

})

})复制代码

这回傻眼了。。。 臭名昭著的 回调地狱 现身了。

更糟糕的是，我们基本上还要对每次请求的结果进行一些处理，代码会更加臃肿，在一个团队中，代码 review 以及后续的维护将会是一个很痛苦的过程。

回调地狱带来的负面作用有以下几点：

代码臃肿。

可读性差。

耦合度过高，可维护性差。

代码复用性差。

容易滋生 bug。

只能在回调里处理异常。

出现了问题，自然就会有人去想办法。这时，就有人思考了，能不能用一种更加友好的代码组织方式，解决异步嵌套的问题。

let 请求结果1 = 请求1();let 请求结果2 = 请求2(请求结果1); let 请求结果3 = 请求3(请求结果2); let 请求结果4 = 请求2(请求结果3); let 请求结果5 = 请求3(请求结果4); 复制代码

类似上面这种同步的写法。 于是 Promise 规范诞生了，并且在业界有了很多实现来解决回调地狱的痛点。比如业界著名的 Q 和 bluebird，bluebird 甚至号称运行最快的类库。

看官们看到这里，对于上面的问题 2 和问题 7 ，心中是否有了答案呢。^\_^

# #### 14、什么是 Promise

Promise 是异步编程的一种解决方案，比传统的异步解决方案【回调函数】和【事件】更合理、更强大。现已被 ES6 纳入进规范中。

代码书写比较

还是使用上面的网络请求例子，我们看下 Promise 的常规写法：

new Promise(请求1)

.then(请求2(请求结果1))

.then(请求3(请求结果2))

.then(请求4(请求结果3))

.then(请求5(请求结果4))

.catch(处理异常(异常信息))复制代码

比较一下这种写法和上面的回调式的写法。我们不难发现，Promise 的写法更为直观，并且能够在外层捕获异步函数的异常信息。

# #### 15、Promise 的常用 API

Promise.resolve(value)

类方法，该方法返回一个以 value 值解析后的 Promise 对象

1、如果这个值是个 thenable（即带有 then 方法），返回的 Promise 对象会“跟随”这个 thenable 的对象，采用它的最终状态（指 resolved/rejected/pending/settled）  
2、如果传入的 value 本身就是 Promise 对象，则该对象作为 Promise.resolve 方法的返回值返回。  
3、其他情况以该值为成功状态返回一个 Promise 对象。

上面是 resolve 方法的解释，传入不同类型的 value 值，返回结果也有区别。这个 API 比较重要，建议大家通过练习一些小例子，并且配合上面的解释来熟悉它。如下几个小例子：

//如果传入的 value 本身就是 Promise 对象，则该对象作为 Promise.resolve 方法的返回值返回。 function fn(resolve){

setTimeout(function(){

resolve(123);

},3000);

}let p0 = new Promise(fn);let p1 = Promise.resolve(p0);// 返回为true，返回的 Promise 即是 入参的 Promise 对象。console.log(p0 === p1);复制代码

传入 thenable 对象，返回 Promise 对象跟随 thenable 对象的最终状态。

ES6 Promises 里提到了 Thenable 这个概念，简单来说它就是一个非常类似 Promise 的东西。最简单的例子就是 jQuery.ajax，它的返回值就是 thenable 对象。但是要谨记，并不是只要实现了 then 方法就一定能作为 Promise 对象来使用。

//如果传入的 value 本身就是 thenable 对象，返回的 promise 对象会跟随 thenable 对象的状态。let promise = Promise.resolve($.ajax('/test/test.json'));// => promise对象

promise.then(function(value){

console.log(value);

});复制代码

返回一个状态已变成 resolved 的 Promise 对象。

let p1 = Promise.resolve(123); //打印p1 可以看到p1是一个状态置为resolved的Promise对象console.log(p1)复制代码

Promise.reject

类方法，且与 resolve 唯一的不同是，返回的 promise 对象的状态为 rejected。

Promise.prototype.then

实例方法，为 Promise 注册回调函数，函数形式：fn(vlaue){}，value 是上一个任务的返回结果，then 中的函数一定要 return 一个结果或者一个新的 Promise 对象，才可以让之后的then 回调接收。

Promise.prototype.catch

实例方法，捕获异常，函数形式：fn(err){}, err 是 catch 注册 之前的回调抛出的异常信息。

Promise.race

类方法，多个 Promise 任务同时执行，返回最先执行结束的 Promise 任务的结果，不管这个 Promise 结果是成功还是失败。 。

Promise.all

类方法，多个 Promise 任务同时执行。  
如果全部成功执行，则以数组的方式返回所有 Promise 任务的执行结果。 如果有一个 Promise 任务 rejected，则只返回 rejected 任务的结果。

# #### 16、如何理解 Promise

为了便于理解 Promise，大家除了要多加练习以外，最好的方式是能够将Promise的机制与现实生活中的例子联系起来，这样才能真正得到消化。

我们可以把 Promise 比作一个保姆，家里的一连串的事情，你只需要吩咐给他，他就能帮你做，你就可以去做其他事情了。  
比如，作为一家之主的我，某一天要出门办事，但是我还要买菜做饭送到老婆单位（请理解我在家里的地位。。）

出门办的事情很重要，买菜做饭也重要。。但我自己只能做一件事。

这时我就可以把买菜做饭的事情交给保姆，我会告诉她：

你先去超市买菜。

用超市买回来的菜做饭。

将做好的饭菜送到老婆单位。

送到单位后打电话告诉我。

我们知道，上面三步都是需要消耗时间的，我们可以理解为三个异步任务。利用 Promise 的写法来书写这个操作：

function 买菜(resolve，reject) {

setTimeout(function(){

resolve(['西红柿'、'鸡蛋'、'油菜']);

},3000)

}

function 做饭(resolve, reject){

setTimeout(function(){

//对做好的饭进行下一步处理。

resolve ({

主食: '米饭',

菜: ['西红柿炒鸡蛋'、'清炒油菜']

})

},3000)

}

function 送饭(resolve，reject){

//对送饭的结果进行下一步处理

resolve('老婆的么么哒');

}

function 电话通知我(){

//电话通知我后的下一步处理

给保姆加100块钱奖金;

}复制代码

好了，现在我整理好了四个任务，这时我需要告诉保姆，让他按照这个任务列表去做。这个过程是必不可少的，因为如果不告诉保姆，保姆不知道需要做这些事情。。（我这个保姆比较懒）

// 告诉保姆帮我做几件连贯的事情，先去超市买菜new Promise(买菜)//用买好的菜做饭

.then((买好的菜)=>{

return new Promise(做饭);

})//把做好的饭送到老婆公司

.then((做好的饭)=>{

return new Promise(送饭);

})//送完饭后打电话通知我

.then((送饭结果)=>{

电话通知我();

})

至此，我通知了保姆要做这些事情，然后我就可以放心地去办我的事情。

请一定要谨记：如果我们的后续任务是异步任务的话，必须return 一个 新的 promise 对象。  
如果后续任务是同步任务，只需 return 一个结果即可。  
我们上面举的例子，除了电话通知我是一个同步任务，其余的都是异步任务，异步任务 return 的是 promise对象。

除此之外，一定谨记，一个 Promise 对象有三个状态，并且状态一旦改变，便不能再被更改为其他状态。

pending，异步任务正在进行。

resolved (也可以叫fulfilled)，异步任务执行成功。

rejected，异步任务执行失败。

Promise的使用总结。

Promise 这么多概念，初学者很难一下子消化掉，那么我们可以采取强制记忆法，强迫自己去记住使用过程。

首先初始化一个 Promise 对象，可以通过两种方式创建， 这两种方式都会返回一个 Promise 对象。

1、new Promise(fn)

2、Promise.resolve(fn)

然后调用上一步返回的 promise 对象的 then 方法，注册回调函数。

then 中的回调函数可以有一个参数，也可以不带参数。如果 then 中的回调函数依赖上一步的返回结果，那么要带上参数。比如

new Promise(fn)

.then(fn1(value）{

//处理value

})复制代码

最后注册 catch 异常处理函数，处理前面回调中可能抛出的异常。

通常按照这三个步骤，你就能够应对绝大部分的异步处理场景。用熟之后，再去研究 Promise 各个函数更深层次的原理以及使用方式即可。

# #### 16、Promsie 与事件循环

Promise在初始化时，传入的函数是同步执行的，然后注册 then 回调。注册完之后，继续往下执行同步代码，在这之前，then 中回调不会执行。同步代码块执行完毕后，才会在事件循环中检测是否有可用的 promise 回调，如果有，那么执行，如果没有，继续下一个事件循环。

关于 Promise 在事件循环中还有一个 微任务的概念（microtask），感兴趣的话可以看我这篇关于nodejs 时间循环的文章 [剖析nodejs的事件循环](https://juejin.im/post/5af1413ef265da0b851cce80" \t "_blank)，虽然和浏览器端有些不同，但是Promise 微任务的执行时机相差不大。

Promise 的升级

ES6 出现了 generator 以及 async/await 语法，使异步处理更加接近同步代码写法，可读性更好，同时异常捕获和同步代码的书写趋于一致。上面的列子可以写成这样：

(async ()=>{

let 蔬菜 = await 买菜();

let 饭菜 = await 做饭(蔬菜);

let 送饭结果 = await 送饭(饭菜);

let 通知结果 = await 通知我(送饭结果);

})();复制代码

是不是更清晰了有没有。需要记住的是，async/await也是基于 Promise 实现的，所以，我们仍然有必要深入理解 Promise 的用法。

# #### 81、Promise是什么，解决了什么，之前怎么实现的

Promise 是异步编程的一种解决方案，比传统的解决方案——回调函数和事件——更合理和更强大。

解决来之前在请求中回调请求产生的回调地狱，使得现在的代码更加合理更加优雅，也更加容易定位查找问题。

# #### 83、promise有几种状态，什么时候会进入catch？

　　三个状态：pending、fulfilled、reject

　　两个过程：padding -> fulfilled、padding -> rejected

当pending为rejectd时，会进入catch

# #### 84、Promise 中reject 和 catch 处理上有什么区别

　　reject 是用来抛出异常，catch 是用来处理异常

　　reject 是 Promise 的方法，而 catch 是 Promise 实例的方法

　　reject后的东西，一定会进入then中的第二个回调，如果then中没有写第二个回调，则进入catch

　　网络异常（比如断网），会直接进入catch而不会进入then的第二个回调

# #### 85、使用class 手写一个promise

[IMG_262](https://www.cnblogs.com/theblogs/p/javascript:void(0);)

//创建一个Promise的类 class Promise{

constructor(executer){//构造函数constructor里面是个执行器

this.status = 'pending';//默认的状态 pending

this.value = undefined//成功的值默认undefined

this.reason = undefined//失败的值默认undefined

//状态只有在pending时候才能改变

let resolveFn = value =>{

//判断只有等待时才能resolve成功

if(this.status == pending){

this.status = 'resolve';

this.value = value;

}

}

//判断只有等待时才能reject失败

let rejectFn = reason =>{

if(this.status == pending){

this.status = 'reject';

this.reason = reason;

}

}

try{

//把resolve和reject两个函数传给执行器executer executer(resolve,reject);

}catch(e){

reject(e);//失败的话进catch }

}

then(onFufilled,onReject){

//如果状态成功调用onFufilled

if(this.status = 'resolve'){

onFufilled(this.value);

}

//如果状态失败调用onReject

if(this.status = 'reject'){

onReject(this.reason);

}

}

}

# #### 87、Promise构造函数是同步执行还是异步执行，那么 then 方法呢？

　　promise构造函数是同步执行的，then方法是异步执行

# #### 88、setTimeout、Promise、Async/Await 的区别

 　事件循环中分为宏任务队列和微任务队列

　　其中setTimeout的回调函数放到宏任务队列里，等到执行栈清空以后执行

　　promise.then里的回调函数会放到相应宏任务的微任务队列里，等宏任务里面的同步代码执行完再执行

　　async函数表示函数里面可能会有异步方法，await后面跟一个表达式

　　async方法执行时，遇到await会立即执行表达式，然后把表达式后面的代码放到微任务队列里，让出执行栈让同步代码先执行