## 如何实现简单的promise

function Promise1(exector) {

this.state = 'pending'; // 状态

this.value = undefined; // 值

this.callbacks = []; // 元素结构 { onResolved() {}, onRejected() {} }

const resolve = value => {

// 状态只能改变一次

if (this.state !== 'pending') return;

this.state = 'resolved';

this.value = value;

if (this.callbacks.length > 0) {

setTimeout(() => {

this.callbacks.forEach(cb => {

cb.onResolved(value);

});

});

}

};

const reject = reason => {

if (this.state !== 'pending') return;

this.state = 'rejected';

this.value = reason;

if (this.callbacks.length > 0) {

setTimeout(() => {

this.callbacks.forEach(cb => {

cb.onRejected(reason);

});

});

}

};

try {

exector(resolve, reject);

} catch (e) {

reject(e);

}

}

Promise1.prototype.then = function (onResolved, onRejected) {

const self = this;

onResolved = typeof onResolved === 'function' ? onResolved : value => value; // 指定返回的 promise 为一个成功状态, 结果值为 value

onRejected =

typeof onRejected === 'function'

? onRejected

: reason => {

throw reason;

}; // 指定返回的 promise 为一个失败状态, 结果值为 reason

const promise = new Promise1((resolve, reject) => {

function handle(callback) {

// 1. 抛出异常 ===> 返回的 promise 变为 rejected

try {

const x = callback(self.value);

// 2. 返回一个新的 promise ===> 得到新的 promise 的结果值作为返回的promise 的结值

if (x instanceof Promise1) {

x.then(resolve, reject); // 一旦 x 成功了,resolve(value), 一旦 x失败了: reject(reason)} else {

// 3. 返回一个一般值(undefined) ===> 将这个值作为返回的 promise 的成功值

resolve(x.value);

} else {

resolve(x);

}

} catch (error) {

reject(error);

}

}

if (this.state === 'pending') {

// 当前 promise 还未确定 pending

// 将 onResolved 和 onRejected 保存起来

self.callbacks.push({

onResolved() {

handle(onResolved);

},

onRejected() {

handle(onRejected);

}

});

}

if (this.state === 'resolved') {

setTimeout(() => {

handle(onResolved);

});

}

if (this.state === 'rejected') {

setTimeout(() => {

handle(onRejected);

});

}

});

return promise;

};

Promise1.prototype.catch = function (onRejected) {

return this.then(null, onRejected);

};

## 面试题1-计算

'11' \* 2 = 22

'a8' \* 3 = NaN

var a = 2, b = 3;

var c1 = a++ + b; // c1 = 5

var c2 = a + ++b; // c2 = 6

## 面试题2-防抖和节流

节流：就是限制一个动作在一段时间内只能执行一次

防抖：就是 当一个动作连续触发，只执行最后一次

/\*\*

\* 防抖

\*

\* @export

\* @param {\*} handle

\* @param {\*} delay

\* @returns

\*/

export function debounce(handle, delay) {

var timer = null;

return function () {

var \_self = this,

\_args = arguments;

clearTimeout(timer);

timer = setTimeout(function () {

handle.apply(\_self, \_args)

}, delay)

}

}

/\*\*

\* 节流

\*

\* @export

\* @param {\*} handler

\* @param {\*} wait

\* @returns

\*/

export function throttle(handler, wait) {

var lastTime = 0;

return function (e) {

var nowTime = new Date().getTime();

if (nowTime - lastTime > wait) {

handler.apply(this, arguments);

lastTime = nowTime;

}

}

}

## 面试题2-对象

var a = {};var b = {key: 'b'};var c = {key: 'c'};var d = [3,5,6];

a[b] = 123;

a[c] = 345;

a[d] = 333;

console.log(a[b]); // 345 { [Object Object]: 123 }

console.log(a[c]); // 345 { [Object Object]: 345 }

console.log(a[d]); // 333 { [Object Object]: 345, “3,5,6”: 333 }

## 不适用循环语句（包括map、forEach方法）实现一个100长度的数组，索引值和值相同的数组[0,1,2,3,4,5........99]

var arr = new Array(100);

//方法一

[...arr.keys()];

//方法二

Array.from(arr.keys());

//方法三

Array.from({length: 100});

// 方法四

借助stringvar arr1 = new Array(101);var str = arr1.join('1,');

str = str.replace(/(1\,)/g, function ($0, $1, index) {

var start = '' + Math.ceil(index/2);

if(index < str.length - 2) {

start += ','

}

return start;

});return str.split(',');

// 方法五（函数式，参考网络）

function reduce(arr, val) {

if(Object.prototype.toString.apply(val)){

return;

}

if(val >= 100) {

return arr;

}

arr.push(val);

return reduce(arr, val+1);

}var res = reduce([], 0)

## 一行代码实现数组去重？

[...new Set([1,2,3,1,'a',1,'a'])]

## 闭包问题

1. 静态(词法)作用域

function foo1(){

var x1=10;

return function bar1(){

console.log('x1= ',x1);

};

}

//"foo"返回一个函数，这个返回的函数使用了自由变量“x1”

var returnedFunction=foo1();

//全局变量“x1”

var x1=20;

//执行已返回的函数

returnedFunction();//10,不是20

//global x2

var x2=10;

//全局函数

function foo2(){

console.log('x2= ',x2);

}

(function(funArg){

//local x2

var x2=20;

//这里是不模糊的，因为我们使用的全局变量x，静态的保存在foo函数的[[Scope]]中，而不是激活函数的调用者作用域中的x

funArg();//10,不是20

})(foo2);

3.

function baz(){

var x=1;

return {

foo:function foo(){return ++x;},

bar:function bar(){return --x;}

};

}

var closures=baz();

console.log('foo() = ', closures.foo())//2

console.log('bar() = ', closures.bar())//1

/\*\* 循环中--------------------------------------------- \*\*/

// 在循环中构建一些函数的迷惑也是和这个特点相关的。在构造的函数中使用循环计数器，一些编程者获得了一些意外的结果，当在函数中使用了同一个计数器时。现在清楚为什么了吧-因为所有的这些函数都有同样的[[Scope]]，在这个[[Scope]]中计数器是最后一次计数值。

var data = [];

for (var k = 0; k < 3; k++) {

data[k] = function () {

alert(k);

};

}

data[0](); // 3, but not 0

data[1](); // 3, but not 1

data[2](); // 3, but not 2

// 这里有一些技巧来解决这些问题。一种技巧就是在作用域链中使用一个附加对象：

var data=[];

for(var k=0;k<3;k++){

data[k]=(function(x){

return function(){

alert(x);

};

})(k)

}

data[0]();//0

data[1]();//1

data[2]();//2

## 让下面的代码可以运行：

const a = [1, 2, 3, 4, 5];

// Implement this

a.multiply();

console.log(a); // [1, 2, 3, 4, 5, 1, 4, 9, 16, 25]

答

const a = [1, 2, 3, 4, 5];

Array.prototype.multiply = function() {

const arr1 = this.map((e) => e \* e)

return this.push(...arr1)

}

// Implement this

a.multiply();

console.log(a); // [1, 2, 3, 4, 5, 1, 4, 9, 16, 25]

## 以下代码会返回 false，解释为什么会这样：

// false

0.2 + 0.1 === 0.3

答

JavaScript中有些数字是用二进制表示的，因为浮点数是不准确的。

+(0.1 + 0.2).toFixed(12) // 0.3

## 题目

1、var a = [];for(let i = 0; i < 10; i++){

a[i] = function () {

console.log(i);

}

}

a[6]();

//输出结果：6//这个题考的匿名函数，//for循环是给每个数组元素赋予相对应的匿名函数：a[0] = function(){console.log(0)}、a[1] = function(){console.log(1)}、a[2] = function(){console.log(2)}...a[9] = function(){console.log(9)};但是此时函数都没有被调用，只有被调用用函数才会执行；//当调用 a[6]()这个匿名函数时，会调用a[6] = function(){console.log(6)}；所以打印的是 6；2、var tmp = new Date();function f() {

console.log(tmp);

if(false){

var tmp = "Hello world";

}

}

f();//输出结果：undefined//这个涉及到变量提升的问题，以上代码相当于function f() {

var tmp;

console.log(tmp);

if(false){

tmp = "Hello world";

}

}//虽然if语句为假，但是tmp变量存在就会提升3、var name = "Hello A!";var objects = {

name : "Hello B!",

getNameFunc : function(){

var that = this;

return function(){

return that.name;

}

}

}let a = objects.getNameFunc()();console.log(a);//输出结果 Hello B;//this作用域指向objects；4.进程和线程的定义区别5.内存分配方式以及它们的区别6.编写一个递归函数，传入两个值，求出最大公约数;

答案：//最大公约数function gcd( x , y){

var max,min,temp;

max = x > y ? x : y ;

min = x < y ? x : y ;

while( max % min ){

temp = max % min;

max = min;

min = temp;

}

return min;

}//通过递归实现function digui(m , n){

return m%n==0?(n):(digui(n,m%n));

}7.编写一个函数，得到字符串中不重复的最大长度；

例如："ababcdcd" 最大不重复字符串为："abcd",长度为4

"qwwekw" 最大不重复字符串为： "wek",长度为3

时间复杂度为O(0)

答案：

var newStr = "";

function f1 (str,indexs = 0){

var tempStr = "";

for(let i=indexs;i<str.length;i++){

if(tempStr.indexOf(str[i]) == -1){

tempStr += str[i];

}else {

if(tempStr.length > newStr.length){

newStr = tempStr;

}

tempStr = '';

f1(str,i);

}

}

}

f1('qwwekw');

console.log('newStr:',newStr.length);

## 内存释放

// 关于内存释放的面试题

function fn() {

var i = 10;

return function (n) {

console.log(n + (++i));

}

}

var f = fn(); //首先把fn执行，然后把执行的结果赋值给f，函数执行只要看函数里面有没有return

f(15); //26 //引用，执行完不会立即释放

f(20); //32

fn()(15); //26 //每次执行完内存释放，作用域销毁

fn()(15); //26

fn()(20); //31

fn()(30); //41

f(30); //43