闭包

能够然我们在外面使用函数里面的变量（局部变量记录在内存中，销毁不掉，要少用）

能够访问另一个函数作用域的变量的函数

function open(){

var m=100;

return function(){

return m=m\*m;

}

}

var n=open();

document.write(n()+"<br>");

document.write(n()+"<br>");

闭包的原理

function open(){

var m=100;

return function(){

return m=m\*m;

}

}

内存泄漏问题

function showId(){

var el=document.getElementById(“app”);

var id=el.id;

el.onclick=fucntion(){

alert(id) //这样会导致闭包引用外层的el，当执行完showId后

//el 无法释放

}

el=null //主动释放el

}

栈是一种先进后出的数据结构

作用域链

变量的使用，从里到外，层层搜索，搜索到了就直接使用

层层搜索

一直到0级作用域，如果还是没有那么就会报错，not definded

function f1(){

var num = 20; // 1级

function f2(){

var num = 30; // 2级

console.log(num);

}

f2();

}

f1();

预解析

就是在浏览器解析代码之前，先把变量的声明，提前到当前作用域的最上面，再把函数的声明提前到作用域的最上面

//变量的提升：var a; window.a=undefined;

if(!(“a”) in window){//=》条件不成立 “a” in window===true

var a=1;

}

console.log(a); //a=undefined

在js非严格模式下

//函数的实参集合与形参变量存在“映射”关系 ：不管其中谁改变了，另外一个都会跟着改变

//在js严格模式下，arguments和形参变量的映射关系呗切断了，相互之间互不干扰

//=>变量提升：var a ; b=aaff11;

var a =4;

function b(x,y,a){

//=>私有作用域:x=1 y=2 a=3

console.log(a); //=>3

arguments[2]=10;//让第三个传递进来的实参等于10

console.log(a);

}

a=b(1,2,3); //undefinded 因为函数b没有返回值

console.log(a);

逻辑与逻辑或 优先级逻辑与高与逻辑或

//=>变量提升：var foo;

var foo=’hello’;

(function(foo){

//私有作用域：foo=’hello’ 但是外面的foo与foor是不同的变量

console.log(foo); //hello

var foo=foo||’word’; //hello // ‘hello’||’word’

console.log(foor); //hello

})(foo);

console.log(foo); //=>把全局foo的值作为实参传递给私有作用域的形参

闭包题目1

//=>变量提升 var arry var res fn()

var arry=[1,2,3,4];

function fn(arry){ //私有arry变量 arry=[1,2,3,4]

arry[0]=0; //私有arry变量 arry=[0,2,3,4]

arry=[0]; //重新开辟了一个数组 arry=[0]

arry[0]=100; //重新开辟了一个数组 arry=[100]

return arry; //返回新数组 arry 值为100

}

var res=fn(arry); //把全局arry存储的值作为实参传递给fn的形参

console.log(arry) ;// 0 2 3 4

console.log(res); //100

闭包题目2

function fn(i){

return function(n){

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

}

}

var f=fn(2); //f输出为fn里面的代码

console.log(f(3));//传入参数为3 类似fn(2)(3)有返回值

console.log(fn(4)(5));// 传入i=4,n=5类似fn(4)(5)

console.log(fn(6)(7));// 传入i=6,n=7类似fn(4)(5)

console.log(f(8));//传入参数为i=2,n=8 类似fn(2)(8)有返回值

闭包题目3

//变量提升 var num;var obj;var fn

var num=10; //window的num

var obj={num:20}; //window的obj

obj.fn=(function(num){ //私有变量num值为20

this.num=num\*3; //this指window即头部num改为60

num++; //私有的num加1

return function(n){

this.num+=n; //this指window即头部num改为60

num++; //私有的num加1

console.log(num); //输出私有的num

}

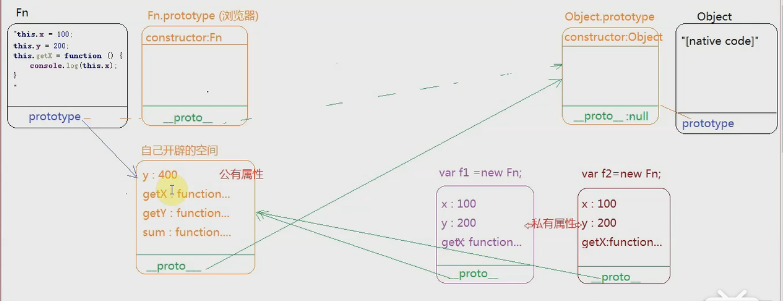
})(obj.num);//对象的值作为实参传递给自定义匿名的形参

var fn=obj.fn;

console.log(fn(5));

console.log(obj.fn(67));

原型链与闭包 题目5



function Fn(){

this.x=100;

this.y=200;

this.getX=function(){

console.log(this.x);

}

}

Fn.prototype={ //开辟一个空间 共有空间

y:400,

getX:function(){

console.log(this.x);

},

getY:function(){

console.log(this.y);

},

sum:function(){

console.log(this.x+this.y);

}

}

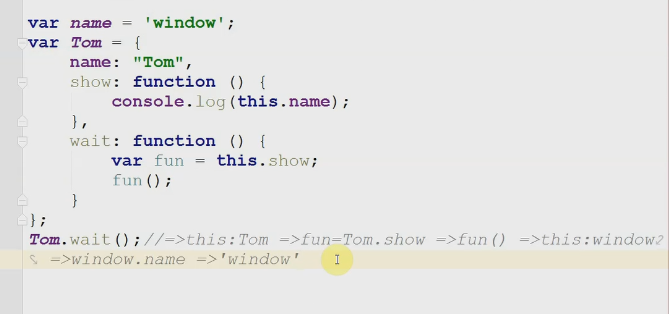
var f1=new Fn();//开辟一个f1空间

var f2=new Fn();//开辟一个f2空间

console.log(f1.getX===f2.getX) //私有不相等

console.log(f1.getY===f2.getY) //共有相等

this前面没点就是window，否则前面是谁就是谁



事件绑定

所有的事件绑定都是异步编程（当触发点击事件，执行方法的时候，循环早已结束）

js同步编程：当前任务没有完成，下面的任务都不会执行，只有等当前任务彻底完成才会执行下面的任务

js异步编程：当前任务没有完成，需要等一会在完成，此时我们可以继续执行下面的

沙箱

在虚拟的环境中，模拟真实世界，但是不会影响真实世界

(function(){}()); //相对自运行匿名函数来说 多了一个 左边括号

eval(‘console.log(“a simple script”);’);

<iframe sandbox=”allow-same-origin” src=”....”></iframe>

|  |  |
| --- | --- |
| 变量 | 全局变量 局部变量 |
| 作用域 | 也就是变量的使用范围 |
| 局部作用域 | 局部变量的使用范围 |
| 全局作用域 | 全局变量的使用范围 |
| 块级作用域 | js里面没有块级作用域 |

自运行匿名函数 常用写法

(function(){})()

!function(){}()

this关键字 this表示当前对象的一个引用

如果单独使用，this表示全局对象

在方法中，this表示此方法所属的对象

在函数中，this表示全局对象

在函数中，在严格模式下，this是未定义的undefined

在事件中，this表示接收事件的元素

原型对象prototype

原型链

实例对象 perseon 指向 person.prototype 指向 Object.prototype 指向 window

面向对象 是一种编程的思维模式，万物皆对象

日期是一个对象 函数是一个对象 数学和正则表达式也是对象

工厂模式

function go(var1,var2){ var jo=new Object(); jo.name=var1;jo.age=var2; return jo}

构造函数 工厂模式的简写

必须new实例化才能出来

构造函数中的this指向正在执行的函数本身，或者指向此函数所属的对象（运行时）

一般函数以大写字母开头的都是构造函数 Object Person Foot

原型函数

Test.prototype.say=function(){ }

argments与...arg

arg是传入不定个参数，在函数内部的arg是个数组

url="https://www.runoob.com/regexp/regexp-intro.html?name=abc&password=123456";

String.prototype.queryStr=function(){

var reg=/([^?!&]+)=([^?!&]+)/g; //正则捕获

var obj={};

this.replace(reg,(...arg)=>{

obj[arg[1]]=arg[2];

});

return obj;

}

console.log("\n");

console.log(url.queryStr());

argments 传入不定个参数

url="https://www.runoob.com/regexp/regexp-intro.html?name=abc&password=123456";

String.prototype.queryStr=function(){

var reg=/([^?!&]+)=([^?!&]+)/g;

var obj={};

this.replace(reg,(arguments)=>{

obj[arguments.split("=")[0]]=arguments.split("=")[1];

});

return obj;

}

console.log("\n");

console.log(url.queryStr());

当str.replace替换中的第一个参数正则，第二个参数是函数时，会发生一个处理效果，把当前正则与字符串进行匹配，有几项匹配的结果，当前这个回调函数就会被触发执行几次。

underfined出现的情况

变量提升，只声明没有定义默认值就是underfined

函数有形参，不过没有传值，形参输出underfined

调用函数，函数没有return返回值，输出underfind

无中生有，使用对象，然而对象没有这个属性 ，属性值就是underfined

严格状态下，没有明确的执行主体，this就是underfined

null出现的情况

手动配置的值或者对象

String.prototype.\_\_proto\_\_为null

null与underfied的区别

null存在，不过值为空

underfied声明了值不存在

1. 判断一个变量是否为数组

var arr=[]

arr instanceof Array //true typeof arry返回一个Object 无法判断是否是数组

instanceof 用来判断引用类型属于哪个构造函数的方法

引用类型 有 数组 对象 函数

1. 原型链继承例子

function animal(val1,val2){

this.name=val1;

this.age=val2;

}

function dog(name,age){

this.type="狗";

// animal.call(this,name,age);

// animal.apply(this,arguments);

// animal.apply(this,[name,age]);

}

dog.prototype=new animal("百灵鸟",100);

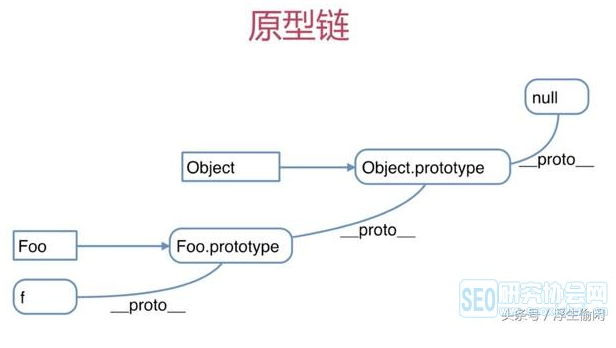
dog.prototype.sex="男";

dog.prototype.constructor=dog //原型从新指向

var m=new dog();

console.log(m)

1. 原型链 \_\_proto\_\_



unction.prototype是引擎创造出来的对象，一开始就有了，又因为其他的构造函数都可以通过原型链找到Function.prototype，Function本身也是一个构造函数，为了不产生混乱，就将这两个联系到一起了

1. 描述new一个对象的过程

通过new一个对象拿到封装函数里面返回的对象 （工厂模式）

创建一个新对象，this指向这个新对象，执行代码，给this赋值，return this

1. 数组去重（乐视tv）
2. document.parentNode和document.parentnode的区别 (腾讯）

document.parentNode文档上一级没有了根节点，具有父节的属性，是null

document.parentnode 这个属性没有所以是underfined

null与underfined的区别，null当前没有，后面可能有，underfined一直没有

1. 怎么规避多人开发函数重名的问题（百度搜索）
2. Javascript如何实现面向对象中的继承？（百度移动）
3. 你理解的闭包作用是什么，优缺点。（乐视）