两大编程思想

面向过程POP

面向对象OOP

**面向对象三大特性：封装、继承、多态**



对象是一组无序的相关属性和方法的集合，所有的事物都是对象

对象由属性和方法组成

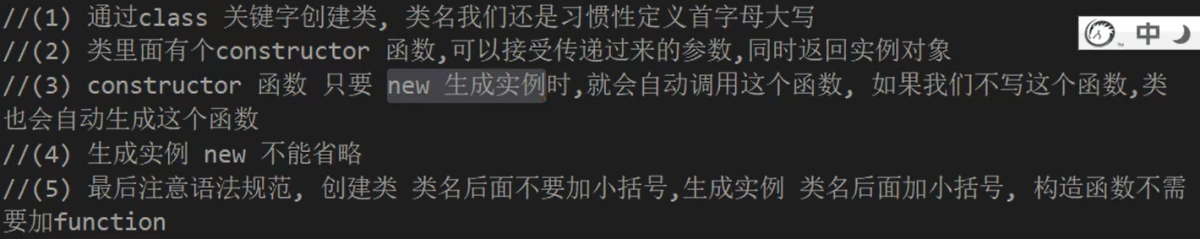
属性：事物的特征

方法：事物的行为

类class 抽象了对象的公共部分

|  |
| --- |
| 以下以ES6为基础  创建类  class name{  // class body  }  利用类创建对象（new）  var mx = new name();  **构造函数constructor（写在类里面）**  constructor(uname){  this.uname = uname;  }  var mx = new name(“叶嘉雄”); |

总结：



类中添加方法（写在类里面）

|  |
| --- |
| say(){  //do something  } |

类的继承

|  |
| --- |
| class A{  constructor(){} //构造  money(){} //普通方法  }  class B extends A{  //继承了父类A  constructor(){  super() // 调用父类构造  }  }  var b = new B();  b.money()  super关键字 调用父类的构造函数  必须写在子类构造函数的第一句    super关键字 调用父类的普通函数 |

ES6的类和对象

1 先有类 才有对象（ES6中类没有变量提升）

2 类里面的共有的属性和方法一定要加this关键字

3 类里面的this指向问题

constructor里面this指向的是实例对象

方法里面的this指向的是调用者

案例——tab切换栏（研究价值大）

构造函数和原型

在ES6之前 JS中没有类的概念

ES6之前，对象不是基于类创建的，而是用一种称为固件函数的特殊函数来定义对象和他们的特征。

创建对象的三种方式：

1对象字面量 ：

var obj = {};

2 new Object()：

var obj = new Object();

**3自定义构造函数**

function Star(uname,age){

this.uname=uname;

this.age=age;

this.sing=function(){

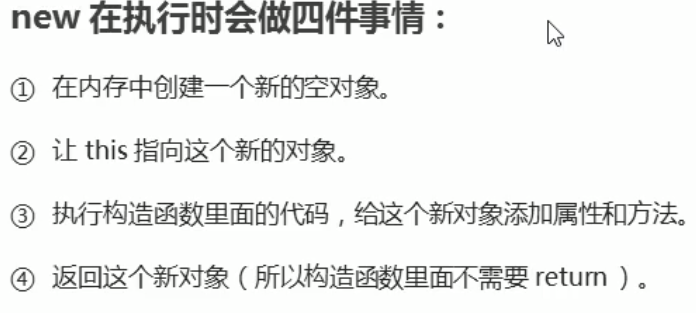
console.log(“sing”);

}

}

var ldh = new Star(“刘德华”,18);

注意：首字母大写 和new搭配使用



静态成员

在构造函数本身上添加的成员（不带this）

例如：Stat.sex=’男’;

只能通过构造函数来访问

实例成员

构造函数内部通过this添加的成员

只能通过实例化的对象来访问

构造函数很好用，但**存在浪费内存的问题**

构造函数原型**prototype**

构造函数通过原型分配的函数是所有对象所**共享的**

JS规定，每一个构造函数都有一个prototype属性，指向另一个对象，这个prototype就是一个对象，这个对象的所有属性和方法，都会被构造函数所拥有的

**我们可以把那些不变的方法，直接定义在prototype对象上，这样所有对象的实例就可以共享这些方法**



原型是一个 对象

原型的作用是 共享方法

一般情况 **公共属性**定义在构造函数里，**公共方法**定义在原型对象身上

对象原型

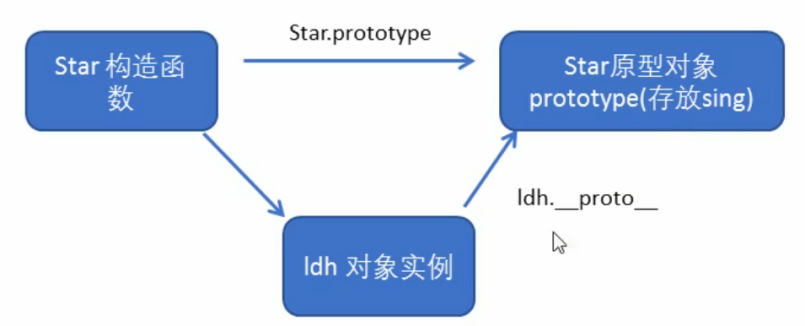
**对象会有一个属性\_\_proto\_\_**指向 构造函数的 prototype原型对象

\_\_proto\_\_对象原型和原型对象prototype是等价的

eg: ldh.\_\_proto\_\_ === Star.prototype // true

方法查找规则：

先从ldh对象身上找，再从构造函数Star的原型对象上找



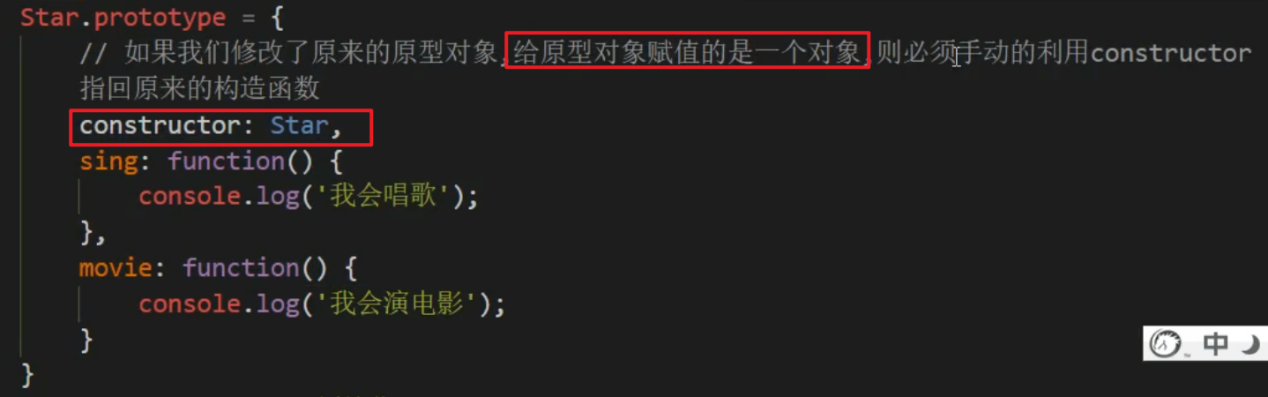
对象原型和原型对象里面都有一个属性：constructor，称为构造函数，指回构造函数本身

ldh.\_\_protp\_\_.constructor === Star // true

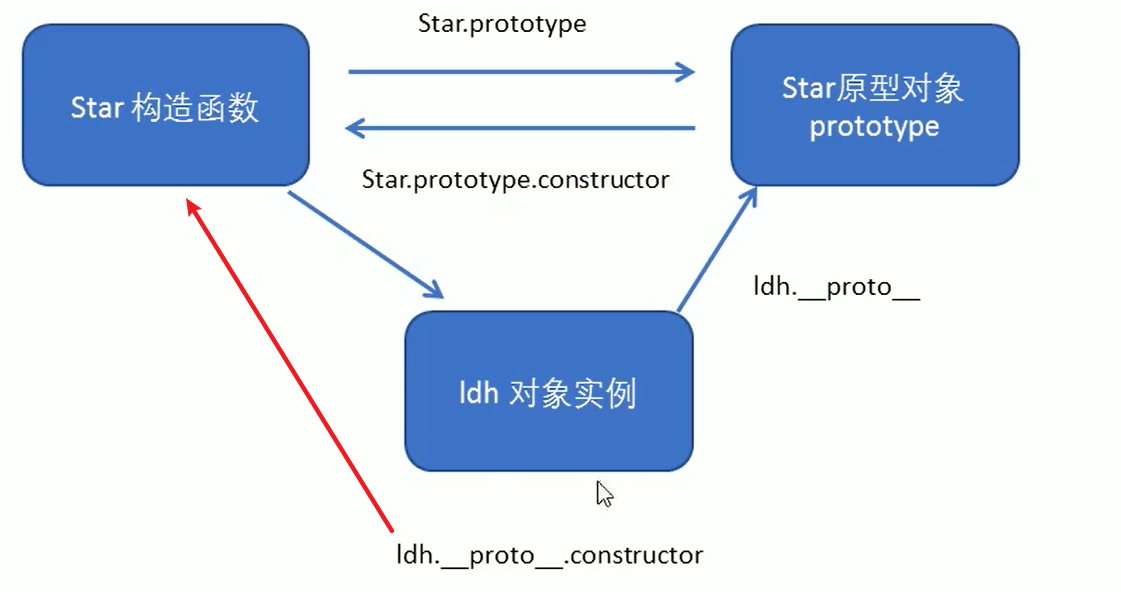
Star.prototype.constructor === Star // true

很多情况下需要手动地利用constructor这个属性指回原来的构造函数

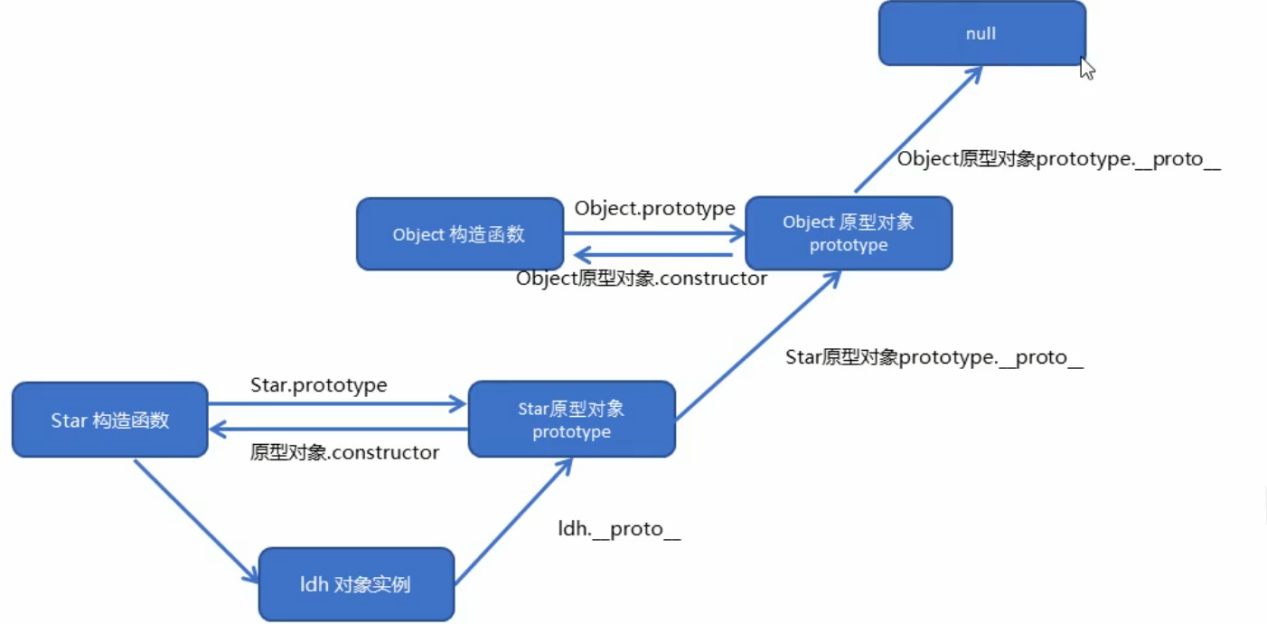
Star.prototype={xx}//这样的形式会重新定义Star的原型对象



构造函数、实例、原型对象三者的关系

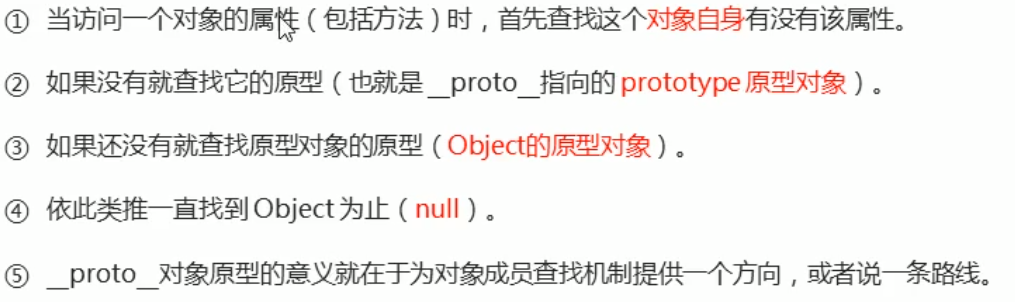


原型链（深刻理解下面的图）



对象查询机制

ldh实例对象-->Star原型对象-->Object原型对象-->null



原型对象this指向

1 构造函数中this指向的是**对象实例（ldh）**

2 构造函数的 原型对象中的函数 中的this指向的是**调用者（一般也是对象实例ldh）**

扩展内置对象

比如给数组添加求和方法

|  |
| --- |
| Array.prototype  Arrty.prototype.sum= function(){  var sum=0;  for(var i=0;i<this.length;i++){  sum+=this.i;  }  return sum;  } |
| var arr1 = [1,2,3];  console.log(arr1.sum()); |

注意：不能写成对象形式

Array.prototype={sum:function(){xxx}} 会覆盖原先的方法

**ES6之前的继承**

ES6之前没有提供extends继承

可以通过**构造函数+原型对象**模拟继承，称为 \*\*组合继承\*\*

**call()方法**

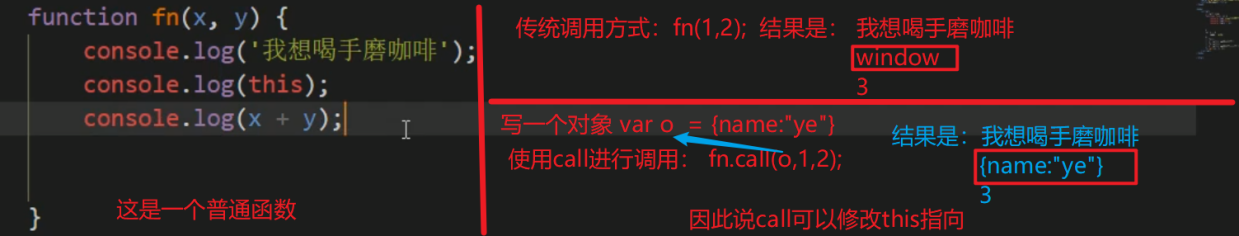
作用：调用这个函数，**并修改函数运行时的this指向**

fun.call(thisArg,[arg1,arg2,....])

thisArg:当前调用函数this的指向对象

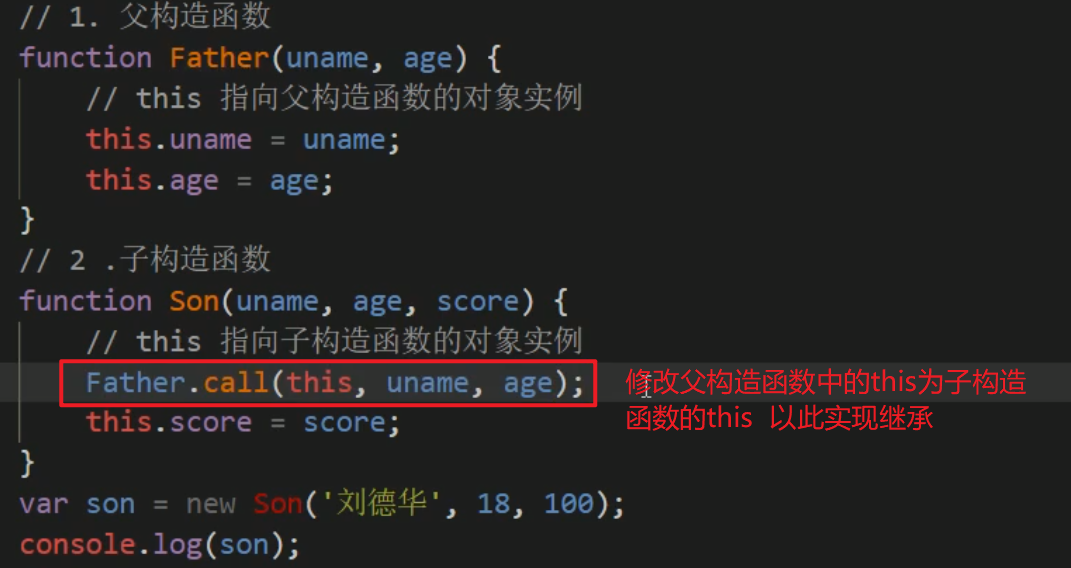
arg1 arg2 传递的参数

例如：



借用构造函数继承父类属性

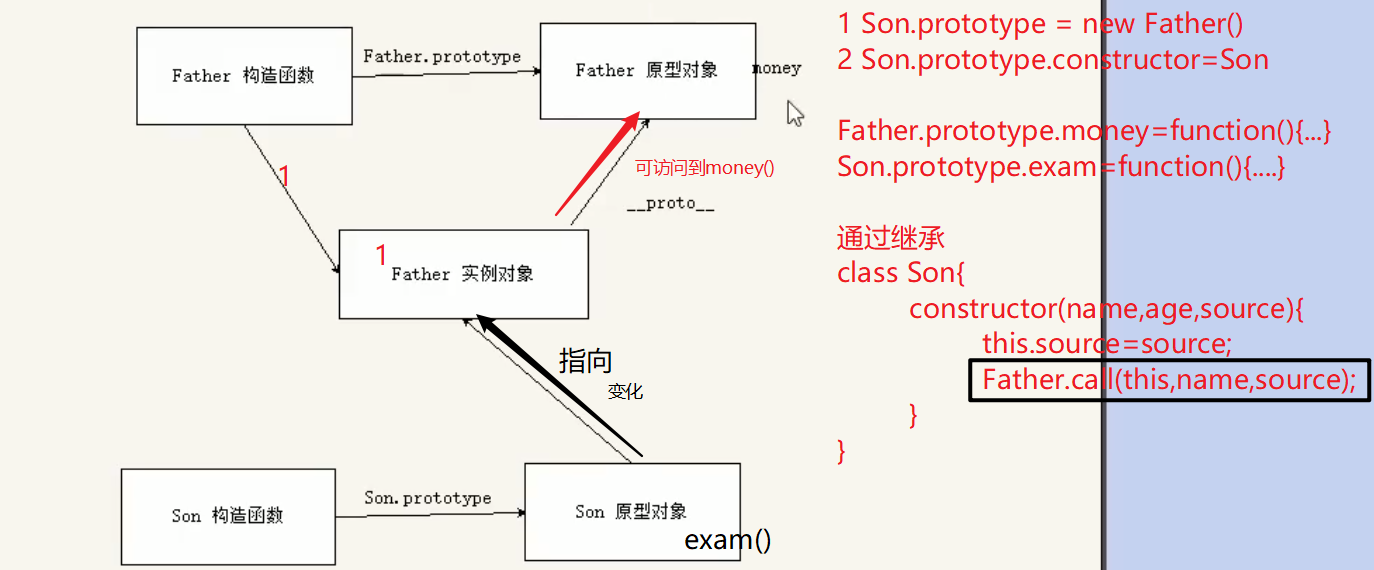
原理：通过call()把父类型的this指向子类型的this



借用构造函数继承父类原型

Son.prototype=new Father();

Son.prototype.constructor=Son;



寄生组合继承：

Son.prototype = Object.create(Father.prototype);

其他操作相同

ES5新增的方法（important）

数组方法

迭代遍历

forEach()、map()、filter()、some()、every()

**\_1 forEach()**

arr.forEach(function(currentValue,index,arr))

currentValue：数组当前值

index： 数组当前索引

arr： 数组对象本身

**2\_filter() 用于筛选 产生新数组**

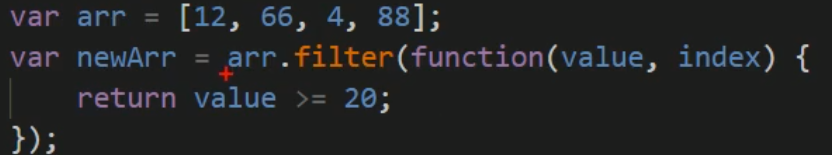
array.filter(function(currentValue,index,arr){})

currentValue：数组当前值

index： 数组当前索引

arr： 数组对象本身

eg：选出数组中大于20的数



**3\_some() 返回布尔值 用于查找是否有 满足条件的元素**

**注意：查找到第一个符合条件的就中止**

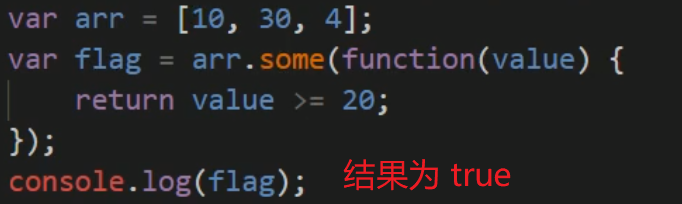
array.some(function(currentValue,index,arr){})

currentValue：数组当前值

index： 数组当前索引

arr： 数组对象本身

eg：数组中有没有大于20的数



案例——查询商品

1 数据用 对象形式存于数组中

2渲染数据使用forEach()



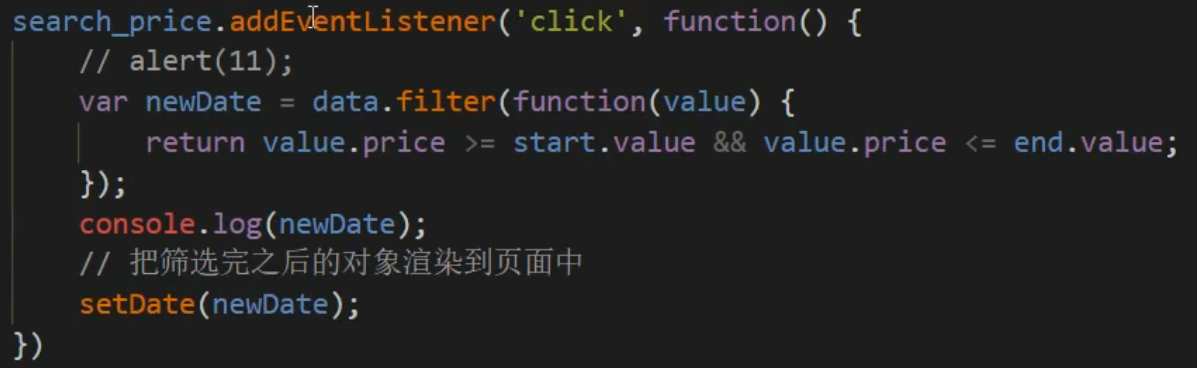
封装成函数

先清空tbody中的内容

创建页面元素createElement()

添加页面元素appendChild()

3根据价格查询产品 使用filter()进行过滤 筛选符合条件的对象



4 按名称查询 使用some()进行查找 （找到唯一的就结束遍历 返回）



**补充**：forEach、filter会遍历到底 Some可以遍历到符合条件的就结束（返回true）

字符串方法

**trim()** 去除左右空格（空白字符）

处理完之后返回新的字符串

不影响原来的，返回新的字符串

对象方法

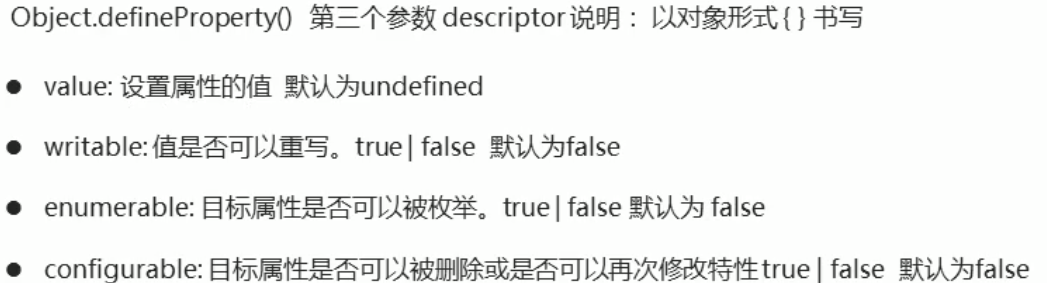
**Object.keys(obj)** 获取对象自身所有的属性名

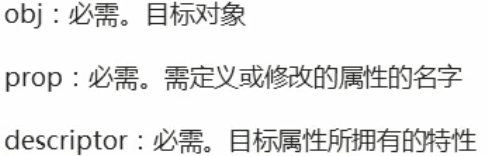
返回由属性名组成的**数组**

效果类似于for...in

**Object.defineProperty()** 定义新属性或修改原有属性







以前的方式：

obj.num=100;

现在的方式：

Object.defineProperty(obj,’num’,{

value:100,

writable:true, #允许重写

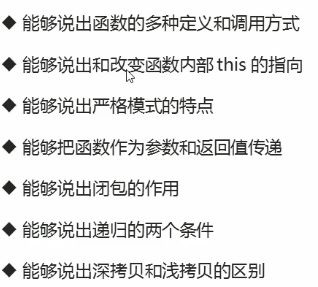
enumerable:true, #允许被枚举

configurable:false #不允许被删除

})

函数进阶

学习目标：



函数定义的方式

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 创建函数的三种方式  1 函数声明方式function关键字（命名函数）  function fn(){}  2 函数表达式（匿名函数）  var fn = function() {}  3 new Function(‘参数1’,’参数2’,’函数体’);  var f = new Function(‘console.log(“123”)’);  var f = new Function(‘a’,’b’,‘console.log(a+b)’);  f(1,2);  注意：参数和函数体都得加引号  所有函数都是Function的实例（对象）    所有的函数也属于对象  调用函数的六种方式  1 普通函数  function fn(){}——fn() / fn.call()  2 对象的方法  var o = {fn:function(){} }——o.fn()  3 构造函数  function Star(){}——new Star()  4 绑定事件函数  btn.onclick = function() {}——点击按钮调用  5 定时器函数  setInterval（function () {},1000）——定时器1秒调1次  6 立即执行函数  ( function(){} )(); —— 自动调用  this指向问题   |  |  | | --- | --- | | 调用方式 | this指向 | | 普通函数调用 | window | | 构造函数调用 | 实例对象 原型对象里的this也是指向这个实例对象 | | 对象方法调用 | 该方法所属的对象 | | 事件绑定方法 | 绑定事件对象 | | 定时器函数 | window | | 立即执行函数 | window | | 箭头函数 | 上下文 |   改变函数内部this指向  1 call方法  主要作用：1调用函数 2改变this指向  fun.call(thisArg,arg1,arg2,....)    call的主要作用是可以实现继承    2 bind方法  返回由指定的this值和初始化参数改造的**原函数拷贝（返回新函数）**  bind不会调用函数，但是能修改函数内部this的指向  fun.bind(thisArg,arg1,arg2,....)    如果有的函数不需要立即调用，又想改变这个函数内部this的指向，此时用bind最合适；不需要改变指向的时候，thisArg写null，  如：当点击一个按钮，就禁用，60s后再开启。  改进前    改进后      3 apply方法  fun.apply(thisArg,[argsArray])  主要作用：1调用函数 2改变this指向  传递的值（参数）必须放在**数组**里面  返回值就是函数的返回值    apply主要应用：借助数学内置对象求数组的最大值最小值    修改this指向方法总结  相同点：都可以修改函数内部的this指向  区别点：call和apply会调用函数；apply传递参数比较特殊（数组），bind不会调用函数  call常用在继承  apply常用在数组有关系的操作  bind不调用函数，可用于改变定时器内部的this指向  严格模式  JS处理提供正常模式，还提供了严格模式，ES5的严格模式是采用具有限制性JS变体的一种方式，即在严格的条件下运行JS代码  严格模式IE10+才支持  严格模式对正常的JS语义做了一些更改  1 消除JS语法的一些不合理、不严谨之处，减少了一些怪异行为  2 消除代码运行的一些不安全之处，保证代码运行的安全  3 提高编译器效率，增加运行速度  4 禁用了ECMAScript的未来版本中可能会定义的一些语法，为未来的JS做好铺垫，如一些保留字不能用作变量名  开启严格模式  可以应用在整个脚本或者个别函数中  因此分为 为脚本开启严格模式 和 为函数开启严格模式 两种情况  1 为脚本开启严格模式  所有语句之前放一个特定的 ： “use strict”      2 为函数开启严格模式    严格模式中的变化  1 变量规定：严禁变量没有声明就赋值；严禁删除已经声明的变量  2 this指向问题：  以前的全局作用域下定义的函数this指向是window，现在则是undefined；  以前构造函数不加new的时候也可以调用，this指向全局对象，在严格模式下必须加new才能改调用；  定时器的this指向还是window；  事件、对象还是指向调用者。  3 函数变化：  不能有重名的参数；  不允许在非函数的代码块下声明函数（不能在if for里写函数）；  更多变化参考MDN  高阶函数  对其他函数进行操作的函数，接收函数作为参数，或者将函数作为返回值输出。        闭包  变量作用域：全局变量、局部变量    什么时候闭包？  闭包是指有权**访问**另一个函数作用域中**变量**的函数。      变量所在的函数，就是闭包函数  闭包的作用    改进一下    闭包的主要作用是：延伸变量的作用范围  闭包案例  1 循环注册点击事件      立即执行函数也称为小闭包  2 循环中的setTimeout()    3 计算打车价格        car.price(5) //13  car.yd(true) // 13+10  思考题  the window 没有闭包的产生  my object 产生了闭包  总结：什么是闭包？闭包的作用  递归  自己调用自己  利用递归求数学题  1 求阶乘 n!  function fn(n){ if(n==1)return 1;else return n\*fn(n-1);}  2 求斐波那契数列  function fbnq(n){ if(n==1||n==2)return 1; else return f(n-1)+f(n-2); }  3 递归遍历数据  略  4 浅拷贝和深拷贝问题  1)浅拷贝    es6新增实现浅拷贝的方法：Object.assign(target,...sources)    2)深拷贝 |

正则表达式

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 概述：Regular Expression 用于匹配字符串中字符组合的模式。在JS中，正则表达式也是对象  作用：验证表单；匹配想要的字符串；提取关键词  匹配，替换，提取  特点：灵活性、逻辑性和功能性极强。简单的方式进行对字符串的复杂控制  实际开发中一般直接复制写好的正则表达式，但是要求能进行一些修改。  创建正则表达式的两种方式  1 RegExp创建  var regexp = new RegExp(/表达式/);  2 字面量创建  var regexp = /表达式/;  检测正则表达式test()  检测字符串是否符合该规则，返回布尔值，参数为测试字符串  regexpObj.test(str);  正则表达式中的特殊字符  称为元字符 如^ \* & 等  1 边界符   |  |  | | --- | --- | | ^ | 开始 | | $ | 结尾 |     2 字符类[]  表示有一系列字符可供选择，只要匹配其中一个就可以了  var rg = /[abc]/; #只要包含a或者b或者c就返回true    var rg = /^[abc]$/; #只有a或者b或者c其中一个    方括号内范围符 [-]  var rg1 = /^[a-z]$/; #26个字母随机一个返回true  字符组合  var rg2 = /^[a-zA-Z0-9\_-]$/;  取反[^]  ^写在[]里  var rg3 = /^[^0-9]$/; #不包含0-9任何一个  3 量词符   |  |  |  | | --- | --- | --- | | \* | 0+ | var reg = /a\*/; | | + | 1+ | var reg = /a+/; | | ? | 0||1 | var reg = /a?/; | | {n} | n | var reg = /a{2}/; | | {n,} | n+ | var reg = /a{3,}/ | | {n,m} | [n,m] | var reg = /a{1,6}/; |   用户验证  var rg2 = /^[a-zA-Z0-9\_-]{6,16}$/;  案例——请输入用户名    括号总结  1 大括号 量词符 里面表示重复次数{}  2 中括号 字符集合 匹配方括号中的任意字符[]  3 小括号 表示优先级  小括号没讲 补充一下  var reg = /^(ab){3}$/; #ababab  var reg = /^ab{3}$/; #abbb  在线测试：<https://c.runoob.com>  预定义类   |  |  | | --- | --- | | \d | [0-9] | | \D | [^0-9] | | \w | [A-Za-z0-9\_] 字母数字下划线 | | \W | [^A-Za-z0-9\_] | | \s | [\t\r\n\v\f] 空格 | | \S | [^\t\r\n\v\f] |   案例——表单验证  正则表达式中的替换replace  stringObject.replace(regexp/substr,replacement)  参数1 被替换的字符串或者正则表达式  参数2 替换为的字符串  返回值是一个替换完毕的字符串  普通用法    正则用法    存在问题：只能替换一个  正则表达式参数  /表达式/[switch]   |  |  |  | | --- | --- | --- | | g | 全局匹配 | global | | i | 忽略大小写 | ignore | | gi | 全局匹配+忽略大小写 |  |   例子 |

ES6

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ES：ECMAScript，由国际标准化组织指定的一种脚本语言的标准规范化  ES6：泛指ES2015及后续的版本  为何使用ES6  新增语法  1 let  用于声明变量的关键字  特点：  1）let声明的变量只在所处于的块级有效（**块级作用域**）    防止内部变量覆盖外部变量  防止循环变量变成全局变量    2）不存在变量提升    必须先声明，再使用   1. 暂时性死区     上述会报错 if里面的tmp先使用，后声明  let经典面试题  1）var练练手      输出两个2  此题关键点在于：变量i是全局的，函数执行时输出的都是全局作用域下的i的值   1. 改成let       输出0 1  关键点：块级作用域里找i的值  2 常量const  作用：声明常量，常量就是值（内存地址）不能变化的量  特点：   1. 具有块级作用域      1. 声明常量时必须赋初值      1. 常量赋值后，值（地址）不能修改       ary[index]=’元素’;并没有修改ary的值（地址）  ary = {‘a’,’b’};就修改了ary的值（地址），因此报错  总结：let、const、var的区别   |  |  |  | | --- | --- | --- | | var | let | const | | 函数作用域 | 块级作用域 | 块级作用域 | | 变量提升 | 不存在变量提升 | 不存在变量提升 | | 值可更改 | 值可更改 | 值不可更改 | |

解构赋值

|  |
| --- |
| ES6中允许从数组中提取值，按照对应位置，对变量赋值。对象也可以实现解构。  1 数组解构      如果解析不成功，变量的值为undefined    2 对象解构    注意：  解构的是对象的属性值，该操作解构接收的名字要和属性名字相同    另一种写法（起别名）    注意：{name:myName,age:myAge}中，冒号左侧的name和age是用于在对象中进行匹配，匹配到了才将相应的值给myName和myAge |

箭头函数

|  |
| --- |
| ES6中新增的定义函数的方式  语法： ()=>{}  const fn = () =>{}//箭头函数赋值给一个变量，变量名即函数名字    特点：  1 函数体中只有一句代码，且代码的执行结果就是返回值，则可以省略大括号{} 和 return    2 如果形参只有一个，则可以省略小括号（）      3 箭头函数不绑定this关键字，箭头函数中的this指向的是函数定义位置的上下文的this    此时箭头函数是在函数fn里定义的，因此它的this就是fn的this  而call方法之后fn的this变成了obj，所以它的this也变成了obj  所以输出的都是obj对象    箭头函数在obj里，obj不可能产生this，因此此时的this是全局作用域下的this（windows），因此是undefined  此时可以在全局作用域下声明一个age,此时就能输出age的值  箭头函数中不能使用arguments |

剩余参数

|  |
| --- |
| 剩余参数语法允许我们将一个不定数量的参数表示为一个数组    function(arg1,arg2,...agrs){}    剩余参数和解构配合使用 |

ES6的内置对象扩展

|  |
| --- |
| 1 Array的扩展方法  1）扩展运算符（展开语法）  作用：将数组或者对象转化为逗号分割的参数序列    为何没有逗号？ console.log（)特殊，把逗号当成分隔符  扩展运算符用于合并数组    扩展运算符 将类数组或可遍历对象转换为真正的数组    就可以调用数组的方法了~  2）构造函数方法：Array.from()  将类数组或者遍历对象转化成真正的数组      该方法可接受第二个参数（函数），作用类似于数组的map方法，用于对每个元素进行处理，将处理后的值放到真正的数组中    3）实例方法find（）  用于找出第一个符合条件的数组成员，如果没有找到就返回undefined  参数是一个函数，代表查询条件    some是查找是否有符合条件的，find是直接找符合的成员   1. 实例方法findIndex()   用于找出第一个符合条件的数组成员的位置，没有找到返回-1     1. 实例方法includes()   表示某个数组中是否包含给定的值，返回bool值    2 String的扩展方法   1. 模板字符串   ES6新增的创建字符串的方式，使用反引号定义    `这是一个模板字符串`  特点：   1. 可以解析变量 ${变量名}        1. 可以换行        1. 可以调用函数${}      1. 实例方法 startsWith()和endsWith   返回布尔值，是否以xx开头和xx结尾     1. 实例方法repeat()   将原字符串重复n次，返回一个新的字符串 |

Set数据结构

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ES6提供的新的数据结构，类似于数组，但是成员都是唯一的，没有重复的成员  作用，如：搜索关键字的存储  Set本身是一个构造函数  创建set对象的两种方式    第二种方式传入的是数组对象  size属性：长度  因此可以这么去掉数组中的重复元素（数组去重）   |  | | --- | | let arr = [1,1,2,3,3];  const s = new Set(arr);  let arr2 = [...s]; |   实例方法   |  |  |  | | --- | --- | --- | | add(value) | 添加 | 返回set本身 | | delete(value) | 删除 | 返回布尔值，代表删除的成功与否 | | has(value) | 是否含有 | 返回布尔值，代表是否含有 | | clear() | 清空 | 无返回值 |     遍历  用forEach方法，没有返回值 |