小程序: WEB问题速查

1. 问题清单: 小程序->首页->JSCORE->day07

2. 高清无码小视频: 小程序->在线->JSCORE->day07

今日扩展视频:

小程序->在线->JSCORE->day07 所有以"作业"开头的视频

不用今天都看完！！！！！！\*\*\*两个月后\*\*\*鄙视面试前\*\*\*看完就行\*\*\*！

3. 单词列表: 小程序->单词->JSCORE->day07

第三阶段上课约定:

1. 上课千万不要跟着记笔记和写代码！

2. 拼写错误！自己找！让我帮你找，你需要手抄20遍，给我拍照！

如何提问:

1. 上课时，案例中，笔记中那句话哪行代码看不懂，立刻微信私信问我！

2. 小程序中问题清单里哪个问题和回答看不懂！立刻微信私信问我！

3. 个人项目中哪个功能不会做！立刻微信私信问我！

4. 刷笔试题时，遇到不会的笔试题！立刻微信私信问我！

下载源代码: 右键->链接另存为！

正课:

1. 箭头函数

2. for of

3. 参数增强

4. 解构

一.箭头函数:

对应小程序视频: 小程序->在线->JSCORE->day07 3. 箭头函数

1. 什么是: 对function的一种简化写法

2. 何时: 今后项目中几乎所有的function都能被箭头函数简写，在将来的新项目中，几乎看不到function的！

3. 如何: 3句话

(1). 几乎所有的function都可以去掉，在()和{}之间加=>

(2). 如果形参列表只有一个形参，则可以省略()

(3). 如果函数体只有一句话，可以省略{}

但是如果这一句话还是return，则必须省略return

4. 示例: 使用箭头函数简写曾经用过的function

1\_arrow.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      // function add(a,b){      //   return a+b;      // }      //var add=function(a,b){ return a+b };      var add=(a,b)=>a+b;      console.log(add(3,5))//8      var arr=[12,3,123,23,2,1];      //arr.sort(function(a,b){return a-b});      arr.sort((a,b)=>a-b);      console.log(arr);      var str="you can you up";      // str=str.replace(/\b[a-z]/ig,function(kw){      //   return kw.toUpperCase()      // });      str=str.replace(/\b[a-z]/ig,kw=>kw.toUpperCase());      console.log(str);      var arr=["亮亮","然然","东东"];      // arr.forEach(function(elem){      //   console.log(`${elem} - 到！`)      // });      arr.forEach(elem=>console.log(`${elem}-到！`));        var arr=[1,2,3,4,5];      // var arr2=arr.map(function(elem){      //   return elem\*2      // });      var arr2=arr.map(elem=>elem\*2);      console.log(arr2);      var arr=[1,2,3,4,5];      // var result=arr.reduce(function(box,elem){      //   return box+elem;      // },0);      var result=arr.reduce((box,elem)=>box+elem,0);      console.log(result);      var n=5;      //var timer=setInterval(function(){      var timer=setInterval(()=>{        console.log(n);        n--;        if(n==0){          console.log("boom!!!");          clearInterval(timer);        }      },1000);      //(function(){      (()=>{        var end=new Date()        console.log(`页面加载完成,at:${end.toLocaleString()}`);      })();    </script>  </body>  </html>  运行结果:  8  (6) [1, 2, 3, 12, 23, 123]  You Can You Up  亮亮-到！  然然-到！  东东-到！  (5) [2, 4, 6, 8, 10]  15  页面加载完成,at:2020/8/4 上午9:41:48  5  4  3  2  1  boom!!! |

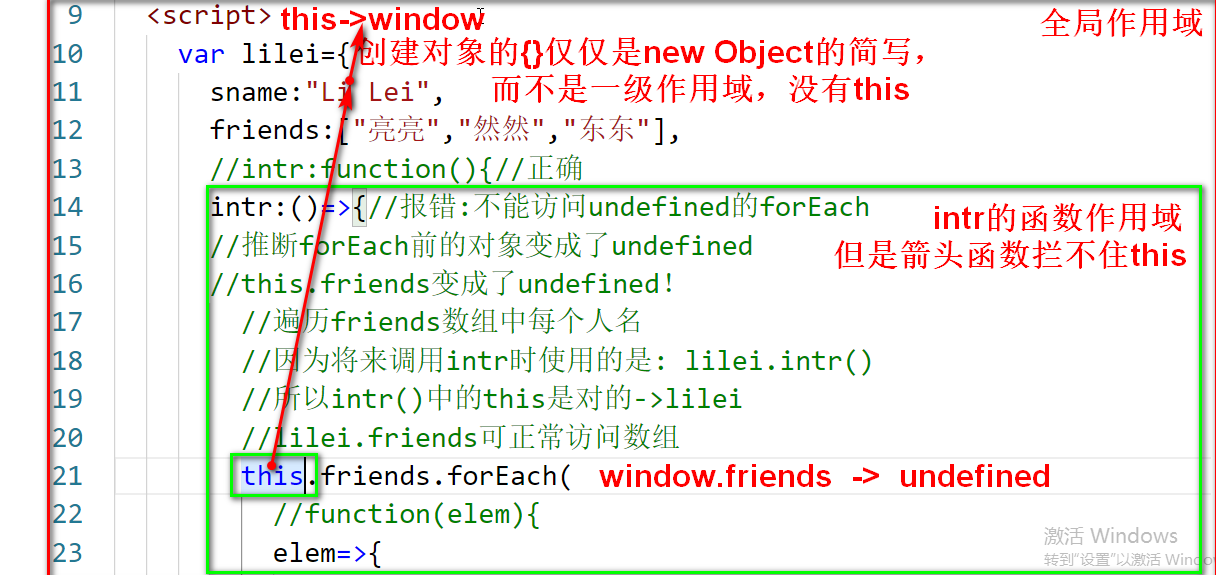
5. 箭头函数特点:

(1). 一旦改为箭头函数，内部的this与外部的this就保持一致(相通)了！

(2). 问题: 不是所有function都能用箭头函数简化







(3). 总结:

a. 可以用: 如果你的函数中根本就没有用到this，或刚好希望内部的this与外部的this保持一致时

b. 不能用: 如果你刚好不希望函数内的this与外部this保持一致时

1). 比如: 对象中的方法，是不能用箭头函数简化的！

2). 解决: ES6中为对象的方法提供了专门的简写！既可简化function，又不影响this！

3). 如何: 今后对象中的方法一律可以简写为：

var 对象={

... : ...,

方法(){ //省略了":function"，不影响this！

}

}

(4). 示例: 使用箭头函数和es6简写来简化function

2\_arrow\_this.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      var lilei={        sname:"Li Lei",        friends:["亮亮","然然","东东"],        //intr:function(){//正确        //intr:()=>{//报错:不能访问undefined的forEach        //推断forEach前的对象变成了undefined        //this.friends变成了undefined！        intr(){//ES6 正确, 不改变原来this的指向          //遍历friends数组中每个人名          //因为将来调用intr时使用的是: lilei.intr()          //所以intr()中的this是对的->lilei          //lilei.friends可正常访问数组          this.friends.forEach(            //function(elem){//错误, 回调函数中的this->window            elem=>{//正确,内部this与外部正确this保持一致了              //回调函数在主函数内被调用时，都是前边既没有点，又没有new，所以绝大多数回调函数中的this->window              //window.sname -> undefined              //改为箭头函数之后，内部的this自动与外部this保持一致（相通）              //内部的this->外部的this->intr点前的lilei              //this.sname->lilei.sname 正确              console.log(`${this.sname} 认识 ${elem}`)            }          )        }      };      lilei.intr();    </script>  </body>  </html>  运行结果:  Li Lei 认识 亮亮  Li Lei 认识 然然  Li Lei 认识 东东 |

二. for of

对应小程序视频: 小程序->在线->JSCORE->day07 4. for of

1. 什么是: 专门用来简化普通for循环的新的for循环

2. 为什么:

(1). 旧的for循环几乎没有可简化的空间了

(2). 而新的forEach是能被数组家的子对象访问，其他类型(类数组对象/字符串)都无法使用

3. 何时: 今后大部分从头到位，只关心元素值不关心元素下标位置的循环都可以用for of来代替普通for循环

4. 如何:

for(var 变量 of 数组/类数组对象/字符串){

//of会依次取出右边数组中的每个元素值，依次放入of前的变量中

}

5. 局限:

(1). of前只有一个变量，只能接元素值，无法获得元素下标位置

(2). of只能从头到尾顺序遍历，不能反向遍历，也不能改变遍历的步长(i+=2, i+=5...)

6. 总结: 因为绝大多数遍历都是从头到尾，一个挨一个的遍历，所以for of的使用场景反而很广泛！所以，绝大多数遍历还是首选for of，只不过个别有特殊需求的遍历，才被迫退一步用for循环。

7. 示例: 使用for of简写for循环

3\_for\_of.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      var arr=["亮亮","然然","东东"];      // for(var i=0;i<arr.length;i++){      //   console.log(`${arr[i]}-到！`)      // }      //arr.forEach(elem=>console.log(`${elem}-到！`))      for(var elem of arr){        console.log(`${elem}-到！`)      }      function add(          ){//计算任意多个数的和      // arguments[          ].length        //          0 1 2 ...        var result=0;        // for(var i=0;i<arguments.length;i++){        //   result+=arguments[i];        // }        //类数组对象，不是数组家人        // arguments.forEach(//报错:arguments.forEach is not a function        //   elem=>result+=elem        // );        for(var elem of arguments){ result+=elem; }        return result;      }      console.log(add(1,2,3));//6      console.log(add(1,2,3,4,5));//15    </script>  </body>  </html>  运行结果:  亮亮-到！  然然-到！  东东-到！  6  15 |

8. vs for in: (暂时不要问)

(1). 回顾for in: 专门遍历自定义下标名称的关联数组或对象的特殊for循环  
 (2). in: 会依次取出in右侧对象中每个属性的属性名

(3). 强调: in不但遍历当前对象中的属性，而且会沿着原型链(\_ \_proto\_ \_属性)继续向上遍历父对象中的没有被enumerable:false隐藏的成员！

(4). 所以: for in通常不会用来遍历索引数组。专门用来遍历关联数组和对象

9. 总结:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | for(var i=0; i<arr.length; i++) | forEach | for of | for in |
| 都是数字下标 | 索引数组 | √ | √ | √ | × |
| 类数组对象 | √ | × | √ | × |
| 字符串 | √ | × | √ | × |
| 自定义下标名称 | 关联数组 | × | × | × | √ |
| 对象 | × | × | × | √ |

三. 参数增强:

对应小程序视频: 小程序->在线->JSCORE->day07 5. 参数增强

1. 参数默认值(default):

(1). 什么是: 即使调用函数时没有传入实参值，对应的形参变量也有默认值可用

(2). 何时: 今后，只要即使调用函数时没有传入实参值，也希望对应的形参变量有默认值可用

(3). 如何: 定义函数时:

function 函数名(形参1, 形参2, 形参n=默认值){

}

(4). 原理:

a. 如果调用函数时，形参变量对应的位置传入了用户自定义的实参值，则优先使用用户自定义的实参值

b. 如果调用函数时，形参变量对应的位置没有传入实参值，则自动启用形参变量上提前准备好的备用默认值

(5). 局限: 只能解决结尾几个形参变量没有实参值的情况，无法解决中间或者开头形参变量缺少实参值的情况。

(6). 解决: 如果希望解决任意一个形参都可缺少实参值的情况，必须用默认值+参数解构共同实现——最后讲  
 (7). 示例:

5\_default.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      function intr(str="主人很懒，什么也没留下"){        console.log(`自我介绍:${str}`)      }      intr("you can you up!");//如果给了实参，就是用用户给的实参。      intr();//如果没给实参，就自动启用默认值使用      //定义一个点餐的函数      function order(zhushi="巨无霸",xiaochi="薯条",yinliao="可乐"){        console.log(`        您本次点的餐包括:          主食:${zhushi},          小吃:${xiaochi},          饮料:${yinliao}        `)      }      //第一个人，三个实参都提供了      order("香辣鸡腿堡","薯条","可乐");      //第二个人: 着急赶火车！一个实参都没提供，也想点餐      order()//希望全都用默认值      //第三个人: 只想换主食为老北京鸡肉卷，别的保持不变      order("老北京鸡肉卷");//后两个传入undefined      //第四个人: 想换主食和小吃，饮料保持默认      order("奥尔良烤腿堡","土豆泥")//最后一个实参传undefined      //第五个人: 只换小吃，主食和饮料保持默认！      //order("菠萝派")//错: 菠萝派给串了！错给了主食      //order(,"菠萝派")//错: 语法不允许前边出现空缺的实参      //第六个人: 只换饮料，前两个保持默认      //order("咖啡")//错: 咖啡错给了主食      //order( , , "咖啡")//错: 语法不允许前边出现空缺的实参      //暂时无法解决——今天最后才能解决！    </script>  </body>  </html>  运行结果:  自我介绍:you can you up!  自我介绍:主人很懒，什么也没留下  您本次点的餐包括:  主食:香辣鸡腿堡,  小吃:薯条,  饮料:可乐    您本次点的餐包括:  主食:巨无霸,  小吃:薯条,  饮料:可乐    您本次点的餐包括:  主食:老北京鸡肉卷,  小吃:薯条,  饮料:可乐    您本次点的餐包括:  主食:奥尔良烤腿堡,  小吃:土豆泥,  饮料:可乐 |

2. 剩余参数(rest)

(1). 什么是: 专门代替arguments来解决不确定实参值个数的问题

(2). 何时: 今后，只要一个函数调用时不确定传入几个实参值时，都用剩余参数语法代替arguments

(3). 为什么: arguments的问题:

a. ES6箭头函数中禁止使用arguments

b. arguments不是纯正的数组类型，无法使用数组家的函数

c. arguments只能获得全部实参值，不能有选择的获得部分实参值

d. 名字好长！

(4). 如何: 定义函数时: ...会自动收集除前几个形参变量外剩余的所有实参值

function 函数名(形参1, 形参2, ...自定义数组名){

//...收集的所有实参值，会自动放在自定义数组中

}

(5). 强调: ES6语法规定...必须放在形参列表的结尾！

(6). 优点:

a. ES6箭头函数中允许使用...arr

b. ...arr是纯正的数组类型，可以使用数组家的函数

c. ...arr可以有选择的获取除之前的形参之外剩余的所有实参值

d. ...arr名字可以自己起，简单

(7). 示例: 使用剩余参数语法代替arguments，并简化代码

6\_rest.hml

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      //定义函数可以计算任意多个数的和      // function add(){//正确      // var add=()=>{//错误      //   var result=0;      //   for(var num of arguments){//报错: arguments is not defined      //     result+=num;      //   }      //   return result;      // }      //          收集      // var add=(...arr)=>{      //   //arr数组中自动保存了所有实参值      //   //既然arr是数组，就可以调用数组家的reduce求和      //   return arr.reduce((box,elem)=>box+elem,0);      //          //复习今天上午的箭头函数      // };      var add=        (...arr)=>arr.reduce((box,elem)=>box+elem,0);      console.log(add(1,2,3))//6      console.log(add(1,2,3,4,5))//15      //定义一个计算总工资的函数      //                   |收集      function jisuan(ename,...arr){        var sum=arr.reduce((box,elem)=>box+elem,0);        console.log(`${ename}的总工资是${sum}`)      }      jisuan("Li Lei",10000,1000,2000);//13000      jisuan("Han Meimei",3000,4000,5000,6000)//18000    </script>  </body>  </html>  运行结果:  6  15  Li Lei的总工资是13000  Han Meimei的总工资是18000 |

3. 打散数组(spread):

(1). 什么是: 专门将一个数组或对象整体打散为多个值，再使用

(2). 问题: apply也能打散数组，但是apply的本职工作是替换this，顺便打散数组！可是有时我们只想打散数组，不想替换this或与this无关！apply就很不好用了！

(3). 何时: 今后只要希望仅打散数组为多个值，再使用时

(4). 如何: 调用函数时:

打散

函数名(...数组名)

(5). 总结: 定义函数时，用...表示收集; 调用函数时, 用...表示打散！

(6). 原理: 先把数组打散为多个值，再将多个值依次分别传给函数的形参变量

(7). 示例:

7\_spead.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      console.log(Math.max(1,7,2,5));//最大值7      console.log(Math.min(1,7,2,5));//最小值1      //问题: 程序中保存多个值，通常都是放在数组中      var arr=[1,7,2,5];      //console.log(Math.max(arr));//NaN      //console.log(Math.min(arr));//NaN      //用apply      //但是本例和this无关！反正替换错了也无所谓      // console.log(Math.max.apply(null,arr));      // console.log(Math.max.apply("",arr));      // console.log(Math.max.apply(arr,arr));      // console.log(Math.max.apply(Math,arr));      //用...      console.log(Math.max(...arr));      console.log(Math.min(...arr));    </script>  </body>  </html>  运行结果:  7  1  7  1 |

(8). 更多...相关的语法糖(简写):

a. 复制一个数组: var arr2=[...arr1];

b. 拼接两个数组以及其他元素: var arr3=[元素1, ...arr1, 元素n, ...arr2 ]

c. 复制一个对象(clone): var obj2={...obj1}

d. 拼接多个对象以及属性: var obj3={ 属性1:值1, ...obj1, 属性n:值n, ...obj2 }

(9). 示例: 使用...语法糖简写实现指定功能:

8\_spread2.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      var arr1=[1,2,3];      //        先打散再放入新数组      var arr2=[...arr1];//[]是new Array()      console.log(arr2);      console.log(arr1==arr2);//false      var arr1=[2,3,4];      var arr2=[6,7,8];      //          先打散再放入新数组      var arr3=[1,...arr1,5,...arr2,9];      console.log(arr3);      var obj1={x:1, y:2};      //        先打散对象，再放入新对象中      var obj2={...obj1};//{}是new Object()的意思      console.log(obj2);      console.log(obj1==obj2);//false      var obj1={x:1,y:2};      var obj2={m:3,n:4};      //              //先打散再放入新对象      var obj3={i:10, ...obj1, j:100, ...obj2};      console.log(obj3);    </script>  </body>  </html>  运行结果:  *(3) [1, 2, 3]*  false  *(9) [1, 2, 3, 4, 5, 6, 7, 8, 9]*  *{x: 1, y: 2}*  false  *{i: 10, x: 1, y: 2, j: 100, m: 3, …}*  i: 10  j: 100  m: 3  n: 4  x: 1  y: 2  \_\_proto\_\_: Object |

四. 解构(destruct):

对应小程序视频: 小程序->在线->JSCORE->day07 6. 解构(上)

1. 什么是解构: 从一个大的对象或数组中仅提取出个别成员单独使用！

2. 何时: 如果一个对象或数组非常大，内容非常多，而每次可能仅仅使用其中个别成员时，都要先将要使用的个别成员单独解构出来，再单独使用！

3. 如何: 3种:

(1). 数组解构:

a. 什么是: 从数组中提取出个别元素值，单独使用

b. 如何: var [变量1, 变量2, ...]=数组

0 1

c. 说明:

1). =左边的[]不是创建数组的意思，仅仅是一种装扮/假扮，将要保存值的变量们都伪装到一个数组结构中，才能骗得=右边数组的信任

2). =右边的数组就会将相同下标位置的元素值，主动赋值给=左边相同位置的变量

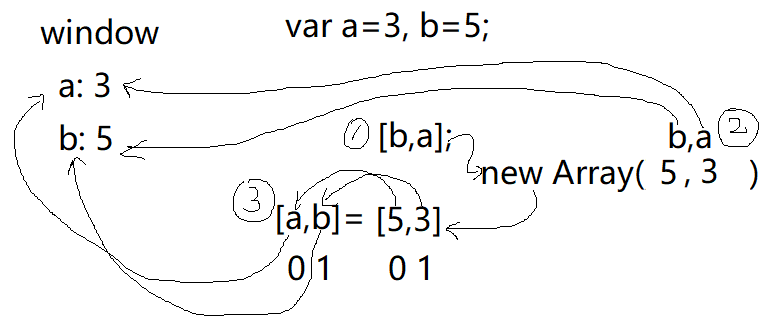
d. 结果: 变量1=数组[0], 变量2=数组[1],...

e. 示例: 使用数组解构，提取出数组中个别值单独使用

9\_destruct\_array.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      var arr=[2020, 8, 4];            //   0   1  2               //年  月 日      //希望从数组中分别解构出年月日单独使用      //var [y,m,d]=arr;      //     0 1 2      //只想要月和日      var [ ,m,d]=arr;      //   0 1 2      // console.log(`年: ${y}`);      console.log(`月: ${m}`);      console.log(`日: ${d}`);    </script>  </body>  </html>  运行结果:  月: 8  日: 4 |

f. 固定套路: 不声明第三个变量，而交换两个变量的值：[a,b]=[b,a];



g. 示例: 不声明第三个变量，交换两个变量的值

9\_destruct\_array.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      var arr=[2020, 8, 4];            //   0   1  2               //年  月 日      //希望从数组中分别解构出年月日单独使用      //var [y,m,d]=arr;      //     0 1 2      //只想要月和日      var [ ,m,d]=arr;      //   0 1 2      // console.log(`年: ${y}`);      console.log(`月: ${m}`);      console.log(`日: ${d}`);        //不声明第三个变量交换两个变量的值      var a=3, b=5;      [a,b]=[b,a];      console.log(a,b);//5,3    </script>  </body>  </html>  运行结果:  月: 8  日: 4  5 3 |

(2). 对象解构:

a. 什么是: 从一个大的对象中提取出个别属性值单独使用

b. 如何:

// 配对儿 配对儿

var {属性名1: 变量1, 属性名2:变量2, ...}=对象

c. 说明:

1). =左边的{}不是创建对象的意思，也只是装扮/伪装，为了赢得=右边对象的信任。

2). =右边的对象，会将自己对应属性名的属性值，自动赋值给=左边的相同属性名对应的变量中

d. 结果: 变量1=对象.属性名1, 变量2=对象.属性名2

e. 问题: 人家原对象中的属性名和方法名已经考虑到了简洁和见名知义，已经很好用了！绝大多数情况下，解构出来的变量名和方法名都与原对象中的属性名和方法名相同。但是，却需要重复写2遍！

f. 解决: 简写: 如果解构时变量名和属性名相同！则只写一个名字即可！一个名字两用！

var { 属性名1, 属性名2, ... }=对象

其中: 属性名1，一个名字两用: 既当做属性名去原对象中配对，又当做变量名准备接住解构出的属性值。

g. 解构失败: 无论数组解构还是对象解构，都存在解构失败的情况

1). 就是: 想要的元素值或属性值，在=右边的数组或对象中没有

2). 解构失败不会报错！而是返回undefined

h. 示例: 使用对象解构提取出对象中个别属性和方法单独使用

10\_destruct\_obj.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      var user={        uname:"dingding",        className:"初一2班",        login(){          console.log(`登录...`)        },        logout(){          console.log(`注销...`)        },        changePwd(){          console.log(`修改密码...`)        }      }      //这次我只想用用户名和注销方法      //   属性名:变量    属性名:变量      //   配对   接值    配对   接值      // var {uname:uname,  logout:logout}=user;      //一个名字两用      //既属性名      //   配对      //又变量名      //   接值      var {uname,logout}=user;      console.log(`用户名:${uname}`);      logout();      //想解构出密码pwd      var {pwd}=user;      console.log(pwd);    </script>  </body>  </html>  运行结果:  用户名:dingding  注销...  undefined |

(3). 参数解构:

对应小程序视频: 小程序->在线->JSCORE->day07 7. 解构(下)

a. 问题: 如果多个实参值都不确定有没有，而且又要求每个实参值必须传给对应的形参变量！

b. 解决: 参数解构:

c. 如何: 2步:

1). 定义函数时: 所有的形参应该定义在一个对象结构中

function 函数名({ //不是创建对象的意思，只是装扮

//属性名: 变量名

//配对 接值

属性名1: 形参名1,

属性名2: 形参名2,

... : ...

}){

... ...

}

2). 调用函数时: 所有实参值也必须放在一个和形参列表相同结构的对象中

函数名({//创建一个对象

//配对 实参值

属性名1: 实参值1,

属性名2: 实参值2,

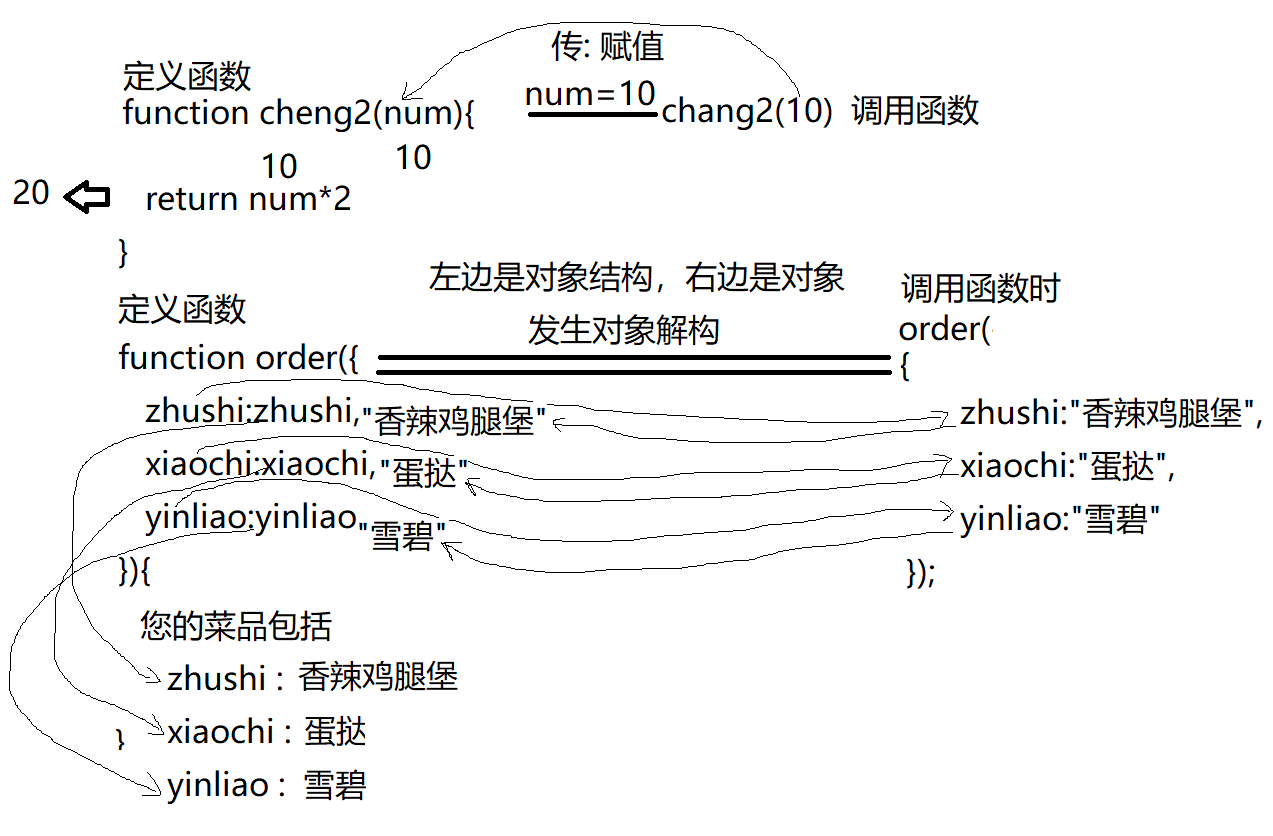
... : ...

})

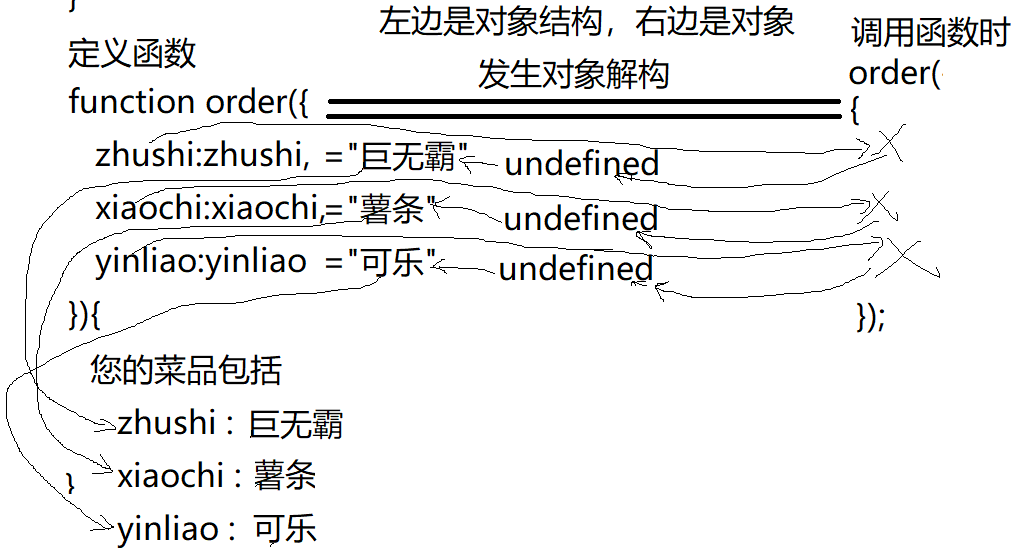
d. 结果: 形参1=实参值1, 形参2=实参值2, ...

e. 原理:

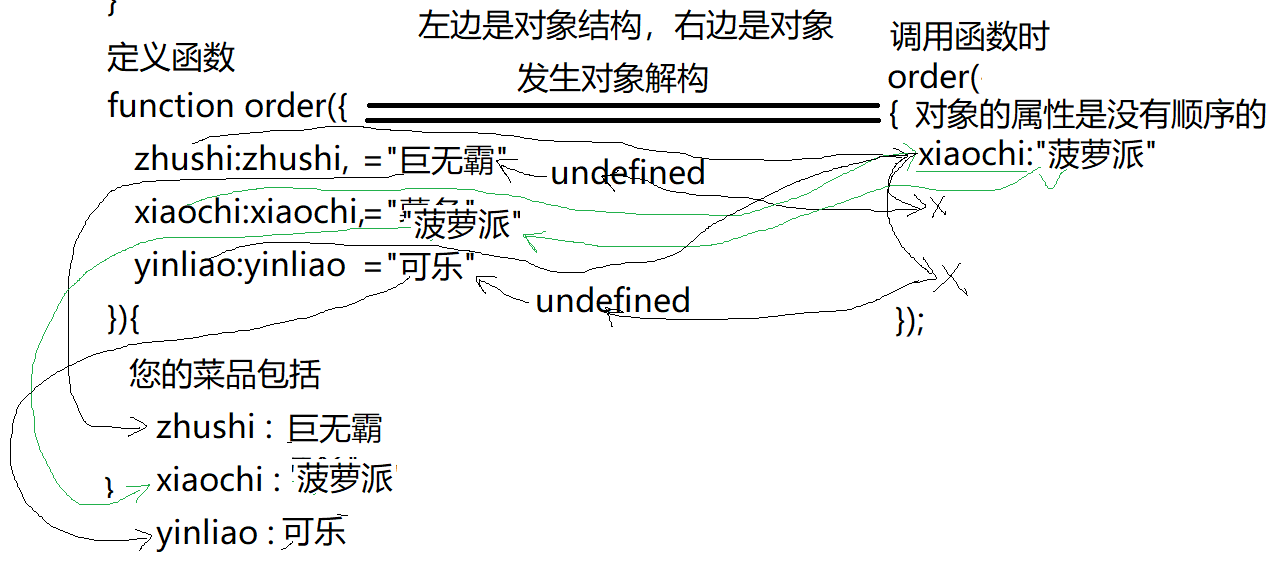
1). 所有实参值都传时



2). 所有实参值都不传时



3). 只传任意一个实参值时



e. 优点: 可以满足所有任意一个实参值不确定的传参情况——灵活

f. 简写: 因为通常用作配对儿的属性名和形参变量名同名，所以，只写一个名称，两用即可！既当做配对的属性名，又当做接实参值的变量名

function 函数名({ //不是创建对象的意思，只是装扮

//一个名字两用:

//既属性名:

// 配对

//又变量名

// 接值

属性名1=默认值1,

属性名2=默认值2,

... : ...

}){

... ...

}

g. 示例: 使用参数解构实现更灵活的函数参数定义

11\_destruct\_params.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      //定义一个点餐的函数      // function order({//装扮成一个对象      // //属性名 : 形参   默认值      // //配对     接值      //   zhushi:zhushi="巨无霸",      //   xiaochi:xiaochi="薯条",      //   yinliao:yinliao="可乐"      // }){      function order({//装扮成一个对象      //一个名字两用:      //既属性名      //   配对      //又形参名      //   接值  默认值        zhushi="巨无霸",        xiaochi="薯条",        yinliao="可乐"      }){        console.log(`        您本次点的餐包括:          主食:${zhushi},          小吃:${xiaochi},          饮料:${yinliao}        `)      }      //第一个人，三个实参都提供了      order({//创建一个对象结构与形参列表对象结构配对        zhushi:"香辣鸡腿堡",        xiaochi:"蛋挞",        yinliao:"雪碧"      });      //第二个人: 着急赶火车！一个实参都没提供，也想点餐      order({})//希望全都用默认值      //第三个人: 只想换主食为老北京鸡肉卷，别的保持不变      order({        zhushi:"老北京鸡肉卷"        //另外两个实参值不写，解构失败，返回undefined，自动启用默认值      });//后两个传入undefined      //第四个人: 想换主食和小吃，饮料保持默认      order({        zhushi: "奥尔良烤腿堡",        xiaochi:"土豆泥"        //第三个实参值不写，解构失败，返回undefined，自动启用默认值      })//最后一个实参传undefined      //第五个人: 只换小吃，主食和饮料保持默认！      //order("菠萝派")//错: 菠萝派给串了！错给了主食      //order(,"菠萝派")//错: 语法不允许前边出现空缺的实参      order({        xiaochi:"菠萝派"        //zhushi和yinliao没有提供实参值，则采用默认值      });      //第六个人: 只换饮料，前两个保持默认      //order("咖啡")//错: 咖啡错给了主食      //order( , , "咖啡")//错: 语法不允许前边出现空缺的实参      order({        yinliao:"咖啡",        //zhushi和xiaochi没有提供实参值，则采用默认值      })      //第七个人: 想换主食和饮料，小吃保持默认      order({        zhushi:"鸡排饭",        yinliao:"鸡蛋汤"        //小吃没有提供实参值，则采用默认值      })    </script>  </body>  </html>  运行结果:  您本次点的餐包括:  主食:香辣鸡腿堡,  小吃:蛋挞,  饮料:雪碧  您本次点的餐包括:  主食:巨无霸,  小吃:薯条,  饮料:可乐    您本次点的餐包括:  主食:老北京鸡肉卷,  小吃:薯条,  饮料:可乐    您本次点的餐包括:  主食:奥尔良烤腿堡,  小吃:土豆泥,  饮料:可乐    您本次点的餐包括:  主食:巨无霸,  小吃:菠萝派,  饮料:可乐    您本次点的餐包括:  主食:巨无霸,  小吃:薯条,  饮料:咖啡    您本次点的餐包括:  主食:鸡排饭,  小吃:薯条,  饮料:鸡蛋汤 |

总结:

8. 替换this: 3种:

(1). 在一次调用函数时，临时替换this，首选:   
 函数.call(对象, 实参值,...)

(2). 临时替换一次this，但是需要打散数组再传参时，被迫改为:

函数.apply(对象, 数组)

(3). 创建一个一模一样的新函数并永久绑定this和部分实参值:

var 新函数名=原函数.bind(对象, 固定实参值, ...)

9. 数组函数:

(1). 判断:

a. 判断数组中是否所有元素都符合要求:

var bool=arr.every(function(value,i,arr){

return 判断条件

})

b. 判断数组中是否包含符合要求的元素:

var bool=arr.some(function(value,i,arr){

return 判断条件

})

(2). 遍历:

a. 单纯简化for循环变量原数组中每个元素:

arr.forEach(function(value,i,arr){

对当前元素执行操作

})

b. 保护原数组不变，返回遍历加工后的新数组

var 新数组=arr.map(function(value, i,arr){

return 加工后的一个新元素值

})

(3). 过滤: 复制出数组中符合要求的元素放入新数组返回

var 新数组=arr.filter(function(value,i,arr){

return 判断条件

})

(4). 汇总: 遍历数组中每个元素，经过求和或其他汇总方式，统计出一个最终结论

var 结果=arr.reduce(function(box,value,i,arr){

return box和value计算出的新临时汇总值

}, 起始值)

10. ES6:

(1). 模板字符串: 今后，只要拼接字符串，都用模板字符串代替+:

a. 整个字符串包裹在一对儿反引号`...`中

b. 反引号``中支持换行、""、''均可使用

c. 反引号中需要动态生成的内容必须放在${}里

d. ${}里:

1). 可以放一切有返回值的合法的变量或js表达式。

2). 不能放程序结构(分支和循环)以及没有返回值的js表达式

(2). let: 今后，声明变量都用let代替var

a. let的好处:

1). 阻止声明提前

2). 让代码块(分支和循环的{})也变成块级作用域，{}块内的变量出了{}无法使用，不会影响外部

b. let的小脾气:

1). 在同一作用域内禁止重复声明;

2). 禁止提前使用;

3). 在全局声明也不保存在window中

(3). 箭头函数: 今后，几乎所有的function都可用箭头函数简写:

a. 如何: 3句话:

1). 去掉function，在()和{}之间加=>

2). 如果只有一个形参，可省略()

3). 如果函数体只有一句话，可省略{}

如果仅有的一句话还是return，必须省略return

b. 今后:

1). 如果函数中没有this或者恰好希望函数内this与函数外this保持一致时，可用箭头函数简写！

2). 如果不希望内外this相同时不能使用箭头函数简写。

(4). for of: 今后只要遍历数字下标的东西，都用for of

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | for(var i=0; i<arr.length; i++) | forEach | for of | for in |
| 数字下标 | 索引数组 | √ | √ | √ | × |
| 类数组对象 | √ | × | √ | × |
| 字符串 | √ | × | √ | × |
| 自定义名称下标 | 关联数组 | × | × | × | √ |
| 对象 | × | × | × | √ |

(5). 参数增强:

a. 参数默认值: 定义函数时最后一个形参不确定有没有实参时

function 函数名(形参1, ..., 最后形参=默认值){

... ...

}

b. 剩余参数: 定义函数时: 只要多个形参不确定，都用剩余参数

function 函数名(其它形参, ...数组名){

//...收集除其它形参外多余的实参值保存到指定数组名的数组中

}

c. 打散数组: 今后调用函数时，只要单纯打散数组再传参时

1). 如何: 函数(...数组)

2). ...口诀: 定义函数时...表示收集, 调用函数时...表示打散

3). ...语法糖:

i. 复制一个数组: var arr2=[...arr1];

ii. 合并多个数组和元素值: var arr3=[...arr1,值,...arr2,值]

iii. 复制一个对象: var obj2={ ... obj1 }

iv. 合并多个对象和属性: var obj3={ ...obj1, 属性:值, ...obj2, 属性:值 }

(6). 解构:

a. 只要想提取出数组中个别元素值，单独使用时，就数组解构:

[变量1, 变量2]=arr;

b. 只要想提取出对象中个别属性值或方法，单独使用时，就对象解构:

var {属性名也是变量名, 属性名也是变量名}=对象

c. 只要多个参数不确定有没有，又要求实参值必须传给指定的形参时，就参数结构:

定义函数时:

function 函数名({

属性名也是形参名="默认值",

属性名也是形参名="默认值",

... = ...

}){

函数体

}

调用函数时:

函数名({

要修改的形参名: 实参值，

... : ... ,

})

总结: this 4种:

1. obj.fun() this->点前的obj对象

2. new Fun() this->new正在创建的新对象

3. 类型名.prototype.共有方法=function(){ ... }

this->将来谁调用这个函数，就指谁

将来调用这个函数的.前的某个子对象

4. fun() 和回调函数 和匿名函数自调 this->默认指window

今日对应小程序视频列表:

小程序->在线->JSCORE->day07 3. 箭头函数

4. for of

5. 参数增强

6. 解构(上)

7. 解构(下)

作业：

1. 复习今日问题清单：小程序->首页->JSCORE->day07

2. (必须)看小程序视频学习ES6中关于对象属性和方法的简写  
 小程序->在线->JSCORE->day07 作业: ES6中关于对象中属性和方法的简写

3. (必须)看小程序视频学习如何使用回调函数让多个异步函数顺序执行

小程序->在线->JSCORE->day07 作业: 如何使用回调函数让多个异步函数顺序执行