准备: 微信小程序: WEB问题速查 小恐龙图标

1. 历届师兄师姐在今天课程中曾经问过的高频的经典的问题清单和解答: 小程序->首页->JSCORE->day06、day07

自己可能没有发现问题，但是想知道别的同学都问过哪些问题

2. 今日课程对应的重点案例的视频以及鄙视面试所需的扩展知识视频：小程序->在线->JSCORE->day06、day07

如果因为某些原因错过直播讲解，不用着急！直接看小程序中视频的回看即可。

如果tmooc不能用了，立刻看小程序视频，小程序视频比tmooc要全，甚至有扩展的新知识。

如果学有余力，将来有能力拿更高薪资，小程序中所有视频必须都要看。但是不必现在都看完。只要在笔试面试前都看完即可！

问题: 个人版小程序有功能限制：禁止在线播放视频

解决: 长按视频连接地址，完整复制视频连接地址和验证码到浏览器打开。通过百度网盘下载后观看。

3. 今日课程中包含的英语单词的翻译：小程序->单词->JSCORE->day06、day07

英文不太好的同学，可以提前看一下第二天需要用到哪些英文单词。

约定: 2件事:

1. 第三阶段禁止上课跟着敲笔记和敲代码！谁抄笔记抄的勤，谁抄代码抄的勤，谁学的越烂！

2. 每个完整功能的案例/视频，至少写三遍: (如果需要画图，必须亲手画图)

(1). 连代码+注释抄一遍 —— 明白这个例子是要做什么事儿

(2). 保留注释，自己试着看着注释，将代码填回来 —— 复习函数和对象

(3). 删除所有代码和注释，自己试着用自己的话，把注释写回来，再翻译为代码！—— 只有自己可以写出人话的注释，这个例子才算自己会！

如果没有问题的同学: （享受vip一对一答疑的时间还剩15天）

1. 上课时那句话没听懂，笔记中那句话看不懂，案例中那句话看不懂，立刻问！

2. 小程序问题清单中哪个问题或解答看不懂，立刻问！

3. 小程序视频中哪个视频看不懂，立刻问

4. 个人项目中哪个功能不会做，立刻问

5. 平时刷笔试题时，遇到不会的笔试题，立刻问

谁问的问题越多！谁的培训费交的越值！

正课:

\*\*\*\*ES5\*\*\*\*

1. 数组新增函数

\*\*\*\*ES6\*\*\*\*

1. 模板字符串

2. let

3. 箭头函数

4. for of

\*\*\*\*ES5\*\*\*\*

一. 数组新增函数:

1. 遍历: 依次取出数组中每个元素，执行相同的操作。2个

(1). forEach: 单纯为了简化普通for循环遍历索引数组

对应小程序视频: 小程序->在线->JSCORE->day06 4. forEach

a. arr.forEach(function(elem,i,arr){

对每个元素值要执行的操作

})

b. 原理:

1). forEach也自带for循环，自动遍历数组中每个元素

2). 每遍历一个元素，就自动调用一次回调函数

3). 每次调用回调函数时，都会自动传入三个实参值:

i. 当前元素值给elem

ii. 当前元素下标位置给i

iii. 当前数组对象给arr

c. 示例: 点名:

1\_forEach.html

|  |
| --- |
| <script>  var arr=["亮亮","然然","东东"];  //依次点名，点到谁，谁喊到！  //用for: 无进一步简化的可能了  // for(var i=0;i<arr.length;i++){  // alert(`${arr[i]} - 到！`);  // }  //用arr.forEach():  // arr.forEach(function(elem,i,arr){  // console.log(`forEach自动调用了一次回调函数, elem=${elem}, i=${i}, arr=${arr}`);  // alert(`${elem} - 到！`);  // })  //arr.forEach(function(elem,i,arr){  //回调函数内容中本次不需要i和arr，可省略  //arr.forEach(function(elem){  //提前尝鲜: 箭头函数=>  // arr.forEach((elem)=>{  // alert(`${elem} - 到！`);  // })  //箭头函数可以更简化: 如果只有一个形参可省略()，如果只有一句话，可省略{}  arr.forEach(elem=>alert(`${elem} - 到！`))  //总结: forEach好处是可以和箭头函数配合有进一步简化的可能！  //关于箭头函数，先不要问为什么！后边再详细讲！  </script>  运行结果: |

(2). map: 依次取出原数组中每个元素值，执行相同的操作后，将修改后的新值放入一个新数组中返回。原数组保持不变。

对应小程序视频: 小程序->在线->JSCORE->day06 6. map原理 数组元素\*2...

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

return 将当前元素值加工后的新值

})

b. 原理:

1). map内部先自动创建一个空的新数组，等待接收加工后的新元素值

2). map也自带for循环，自动遍历数组中每个元素

3). 每遍历一个元素就自动调用一次回调函数

4). 每次调用回调函数时，都自动传入三个实参值:

i. 当前元素值给elem

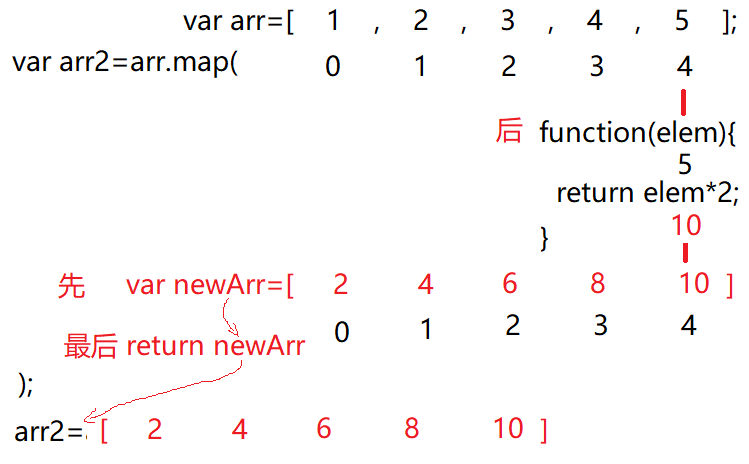
ii. 当前元素下标位置给i

iii. 当前数组对象给arr

5). 回调函数内部会对本次取出的当前元素值进行加工，并返回加工后个一个新元素值

6). map会将回调函数本次返回的新值，保存到新数组中相同的位置上。

7). 遍历结束后，map会将包含修改后的新元素值的新数组返回



c. 示例: 将原数组中每个元素值\*2，并放入新数组中返回

2\_map.html

|  |
| --- |
| <script>  var arr=[1,2,3,4,5];  //想对数组中每个元素\*2后，放入一个新数组返回，保护原数组不变  var arr2=arr.map(function(elem){  return elem\*2;  });  console.log(arr);  console.log(arr2);  </script>  运行结果:  [1, 2, 3, 4, 5]  [2, 4, 6, 8, 10] |

2. 过滤: 复制出原数组中符合条件的元素值，放入新数组中返回，原数组中的元素保持不变，不会少元素！

对应小程序视频: 小程序->在线->JSCORE->day06 7. filter

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

return 判断条件

})

b. 原理:

1). filter内部先自动创建一个空的新数组，等待接收加工后的新元素值

2). filter也自带for循环，自动遍历数组中每个元素

3). 每遍历一个元素就自动调用一次回调函数

4). 每次调用回调函数时，都自动传入三个实参值:

i. 当前元素值给elem

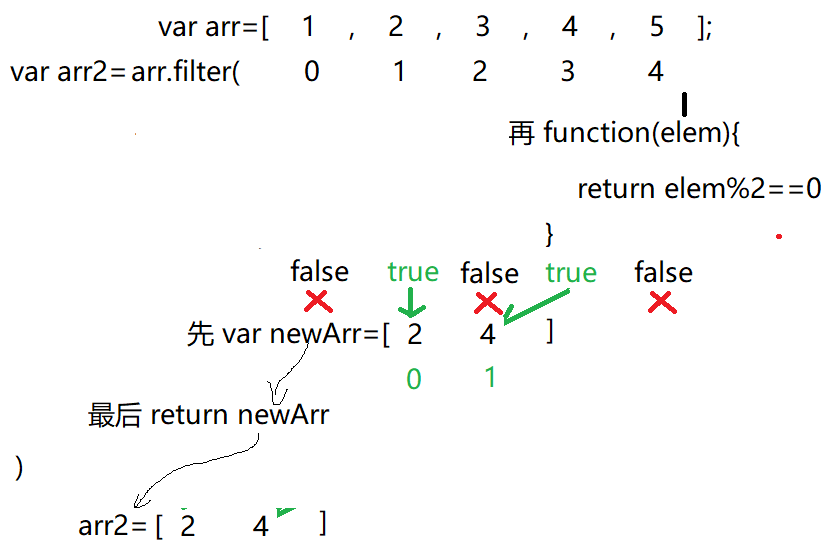
ii. 当前元素下标位置给i

iii. 当前数组对象给arr

5). 回调函数内部会判断当前元素值是否符合要求，并返回判断结果

6). 如果本次判断结果为true，则filter自动将当前元素追加到新数组中。如果本次判断结果为false，则filter什么也不干！

7). 遍历结束后，filter会将包含符合条件的元素的新数组返回



c. 示例: 复制出数组中的偶数，放入新数组中返回

3\_filter.html

|  |
| --- |
| <script>  var arr=[1,2,3,4,5];  //想复制出数组中的偶数组成新数组  var arr2=arr.filter(function(elem){  return elem%2==0  });  console.log(arr);  console.log(arr2);  </script>  运行结果:  [1, 2, 3, 4, 5]  [2, 4] |

3. 汇总: 对数组中所有元素求和

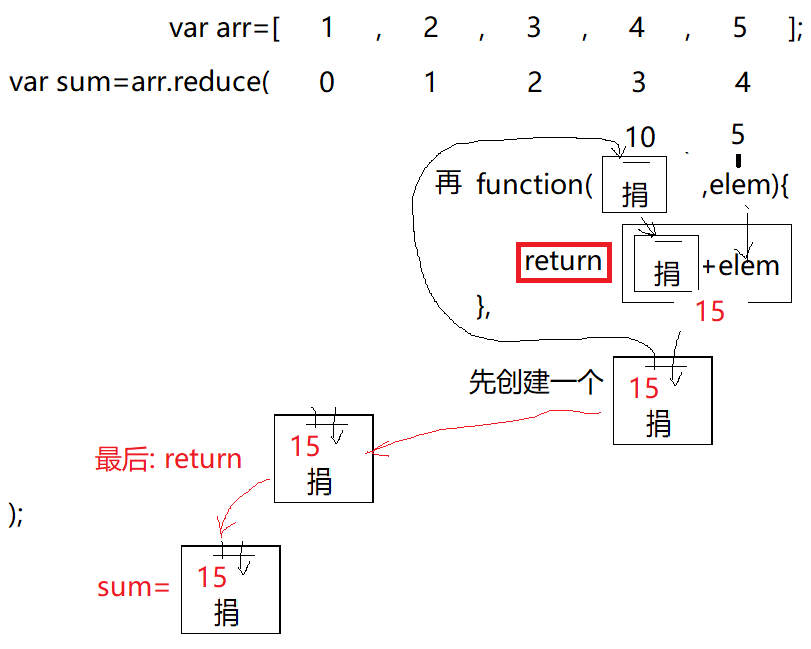
对应小程序视频: 小程序->在线->JSCORE->day06 8. reduce原理 简单 数组元素求和

a. var 汇总结果=arr.reduce(function(捐款箱, elem, i, arr){

return 旧捐款箱中的值加上当前新元素值获得的新汇总值，自动放入捐款箱中

},捐款箱中的起始值)

b. 原理:



1). reduce函数内，先创建一个变量，暂且称为捐款箱，保存reduce函数的第二个实参值——汇总的起始值。

2). reduce函数内也自带for循环，自动遍历数组中每个元素

3). 每遍历到一个元素就自动调用一次回调函数

4). 每次调用回调函数时，都会自动传入四个值:

i. 捐款箱: 当前reduce函数内保存的捐款箱变量中的临时汇总值

ii. elem: 当前新元素值

iii. i: 当前遍历到的位置

iv. arr: 当前数组对象

5). 在回调函数内部，要将新遍历到的当前元素值和捐款箱变量中的临时汇总值相加得到新的临时汇总值，并将新汇总值返回给reduce函数

6). reduce函数每次收到回调函数返回的新临时汇总值，都重新保存回捐款箱变量中，为继续累加下一个元素值做准备。

7). 当所有元素都遍历完，reduce就返回最后捐款箱变量中的汇总值，就是这个数组所有元素的和。

c. 示例: 对一个数组中的元素求和

4\_reduce.html

|  |
| --- |
| <script>  var arr=[1,2,3,4,5];  //相对数组中的元素值求和  var sum=arr.reduce(function(捐款箱,elem){  console.log(`自动调用一次回调函数:捐款箱中现有${捐款箱}元, 当前新元素值${elem}元，现在捐款箱中的钱+当前新元素值=${捐款箱+elem}元，还放回捐款箱，为继续累加下一个元素值做准备.`)  return 捐款箱+elem  },0);//捐款箱中的钱数从0开始累加  console.log(sum);  </script>  运行结果:  自动调用一次回调函数:捐款箱中现有0元, 当前新元素值1元，现在捐款箱中的钱+当前新元素值=1元，还放回捐款箱，为继续累加下一个元素值做准备.  自动调用一次回调函数:捐款箱中现有1元, 当前新元素值2元，现在捐款箱中的钱+当前新元素值=3元，还放回捐款箱，为继续累加下一个元素值做准备.  自动调用一次回调函数:捐款箱中现有3元, 当前新元素值3元，现在捐款箱中的钱+当前新元素值=6元，还放回捐款箱，为继续累加下一个元素值做准备.  自动调用一次回调函数:捐款箱中现有6元, 当前新元素值4元，现在捐款箱中的钱+当前新元素值=10元，还放回捐款箱，为继续累加下一个元素值做准备.  自动调用一次回调函数:捐款箱中现有10元, 当前新元素值5元，现在捐款箱中的钱+当前新元素值=15元，还放回捐款箱，为继续累加下一个元素值做准备.  15 |

\*\*\*\*ES6\*\*\*\*

一. 模板字符串

对应小程序视频: 小程序->在线->JSCORE->day07 1. 模板字符串

1. 什么是: 支持换行以及动态生成内容的特殊字符串

2. 为什么: 旧js中，用+拼接字符串，极容易和算术计算的+法计算混淆

3. 何时: 今后，只要动态拼接字符串内容时，都用模板字符串代替传统的用+拼接

4. 如何:

(1). 整个模板字符串必须用一对儿反引号`包裹——键盘左上角ESC键正下方那个键

(2). 在模板字符串中支持换行

(3). 在模板字符串如果包含用js表达式动态生成的内容，必须用${}包裹

5. 结果: js程序会先将${}中的js表达式结果计算出一个子字符串，然后再将子字符串拼接到原字符串中${}位置代替。结果，模板字符串其实还会变为一个普通的字符串。

6. 总结: ${}内都可以放哪些东西？不能放哪些东西？和Vue框架中的{{}}要求是一样的！

(1). 可以放: 变量，运算，三目运算，有返回值的函数调用，创建对象，访问数组元素

(2). 不能放:

a. 程序结构(分支和循环)

b. 没有返回值的函数调用或js表达式

7. 示例: 使用模板字符串，动态生成字符串内容:

5\_templateString.html

|  |
| --- |
| <script>  var price=12.5; //单价  var count=5; //数量  //打印一张简单的超市小票  console.log(`  单价: ¥${price.toFixed(2)}  数量: ${count}  ------------------------------  小计: ¥${(price\*count).toFixed(2)}  `);  var sex=1;//性别: 男  var orderTime=1591338064518;//下单时间:2020/6/5 14:21  var week=["日","一","二","三","四","五","六"];  // 0 1 2 3 4 5 6  console.log(`  性别:${sex==1?"男":"女"},  下单时间:${new Date(orderTime).toLocaleString()},  今天星期${week[new Date().getDay()]}  `)// 5  // week[5]  // 五  </script>  运行结果：  单价: ¥12.50  数量: 5  ------------------------------  小计: ¥62.50    性别:男,  下单时间:2020/6/5 下午2:21:04,  今天星期五 |

二. let

对应小程序视频: 小程序->在线->JSCORE->day07 2. let

1. 什么是let: ES6中专门代替var，用来声明变量新关键字

2. 为什么: var的缺点:

(1). var会被声明提前——破坏了程序的执行顺序

(2). var声明的变量没有块级作用域——导致块内的变量会提升到块外，影响外部代码

3. 示例: var的缺点:

6\_var.html

|  |
| --- |
| <script>  var t=0;//定义一个全局变量，用来累加以下两个函数的执行时间  function fun1(){  console.log(`fun1耗时0.3s`);  t+=0.3;  }  function fun2(){//是作用域  //var t; //undefined  //从此: 因为fun2中有了局部变量t，就不会再用全局的t了  console.log(`fun2耗时0.8s`);  t+=0.8;//只是加在局部变量t上  //后期维护过程中，又新增了一段代码:  if(false){//不是作用域 //根本不会执行  var t=new Date();  console.log(`当前时间: ${t.toLocaleString()}`)  }//虽然这段代码没有执行，但是却让原本正常执行的其它程序，发生错误！  }  fun1();  fun2();  console.log(`共耗时${t}s`);//1.1  </script>  运行结果:  fun1耗时0.3s  fun2耗时0.8s  共耗时0.3s |

4. 何时: 今后新的框架程序中都是用let代替var的

5. 优点: 2个:

(1). let不会被声明提前——保证程序顺序执行

(2). let为js添加了块级作用域的效果——防止块内的变量，影响到块外部。

6. 原理: let做了两件事:

(1). let相当于匿名函数自调

(2). let还可能自动修改变量名

7. 示例: 使用let代替var

7\_let.html

|  |
| --- |
| <script>  var t=0;//定义一个全局变量，用来累加以下两个函数的执行时间  function fun1(){  console.log(`fun1耗时0.3s`);  t+=0.3;  }  function fun2(){//是作用域  console.log(`fun2耗时0.8s`);  t+=0.8;//只是加在局部变量t上  //后期维护过程中，又新增了一段代码:  if(true){//变成作用域了  //let 让if后{也变成了作用域  let t=new Date();  console.log(`当前时间: ${t.toLocaleString()}`);  //在底层已经被翻译为以下代码:  // (function(){  // var \_t=new Date();//成了if内的局部变量，不会影响外部  // console.log(`当前时间: ${\_t.toLocaleString()}`)  // })();  }//就算if执行，里边的let t和外边的t，不会互相影响  }  fun1();  fun2();  console.log(`共耗时${t}s`);//1.1  </script>  运行结果:  fun1耗时0.3s  fun2耗时0.8s  当前时间: 2020/6/5 下午3:23:21  共耗时1.1s |

8. let的三个小脾气:

(1). 即使在全局声明的let，也不会保存到window中！

(2). 在let之前禁止提前使用该变量！

(3). 在同一作用域内，不允许反复let同名的变量

9. 示例: let的小脾气

8\_let2.html

|  |
| --- |
| <script>  var a=10;  console.log(window.a);//10  //(function(){  let b=10;  console.log(window.b);//undefined  //})()  console.log(c);//undefined  var c=10;  console.log(c);//10  //console.log(d);//报错: Cannot access 'd' before initialization  // 不能 访问 的 在 初始化 之前  //let d=10;  //console.log(d);//10  var e=10;  var e=100;  console.log(e);//100  //let f=10;  //let f=100; //报错：Identifier 'f' has already been declared  // 标识符/变量名 f 已经 被 声明过了  </script>  运行结果:  10  undefined  undefined  10  100 |

三. 箭头函数

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

1. 什么是箭头函数: 对旧js中function的简写

2. 为什么: 简单！

3. 何时: 今后几乎所有的function都可用箭头函数简化——在将来框架开发中几乎看不到任何function了！

4. 如何: 3句话口诀

(1). 去掉function在()和{}之间加=>

(2). 如果()中只有一个形参，可省略()，但是如果()中没有形参，则()不能省略

(3). 如果函数体只有一句话，可省略{}和这条语句结尾的;

强调: 如果仅有的一句话还是return，则必须省略return——语法规定，没有为什么

5. 示例: 使用箭头函数简写各种function

9\_arrow.html

|  |
| --- |
| <script>  //function add(a,b){  //var add=function(a,b){  var add=(a,b)=>//{  console.log(`${a}+${b}=${a+b}`)  //}  add(3,5);  var arr=[12,3,123,32,2,1];  //arr.sort(function(a,b){ return a-b; });  arr.sort((a,b)=>a-b)  console.log(arr);  var str="you can you up";  // str=str.replace(/\b[a-z]/g, function(keyword){  // return keyword.toUpperCase();  // })  str=str.replace(/\b[a-z]/g,kw=>kw.toUpperCase())  console.log(str);  var arr=["亮亮","然然","东东"]  //arr.forEach(function(elem){  arr.forEach(elem=>/\*{\*/  console.log(`${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(arr);  console.log(arr2);  var arr=[1,2,3,4,5];  // var sum=arr.reduce(function(box,elem){  // return box+elem  // },0);//从0开始累加  var sum=arr.reduce((box,elem)=>box+elem,0)  console.log(sum);  //(function(){  (()=>{  console.log("我是匿名函数自调");  console.log(`当前系统时间:${new Date().toLocaleString()}`)  })();  var t=5;  //var timer=setInterval(function(){  var timer=setInterval(()=>{  if(t>0){//如果倒计时没结束  console.log(t);//就输出倒计时  t--;//倒计时-1  }else{//否则如果倒计时已经结束  console.log(`boom!!!`);//就爆炸  clearInterval(timer);//并停止定时器  }  },1000)  </script>  运行结果:  3+5=8  [1, 2, 3, 12, 32, 123]  You Can You Up  亮亮 - 到  然然 - 到  东东 - 到  [1, 2, 3, 4, 5]  [2, 4, 6, 8, 10]  15  我是匿名函数自调  当前系统时间:2020/6/5 下午4:12:28  5  4  3  2  1  boom!!! |

6. 箭头函数的特征: 箭头函数内的this与箭头函数外的this始终保持一致.

强调: 这个特点仅作用域this，其他变量的作用域保持不变！

7. 总结: 何时可以使用箭头函数，何时不能用箭头函数:

(1). 可以使用箭头函数:

a. 如果函数中根本就没有用到this，可随意用箭头函数简写

b. 如果函数中用到了this，但恰好希望函数内的this与函数外的this保持一致，则可以用箭头函数箭头

(2). 不能用箭头函数简写:

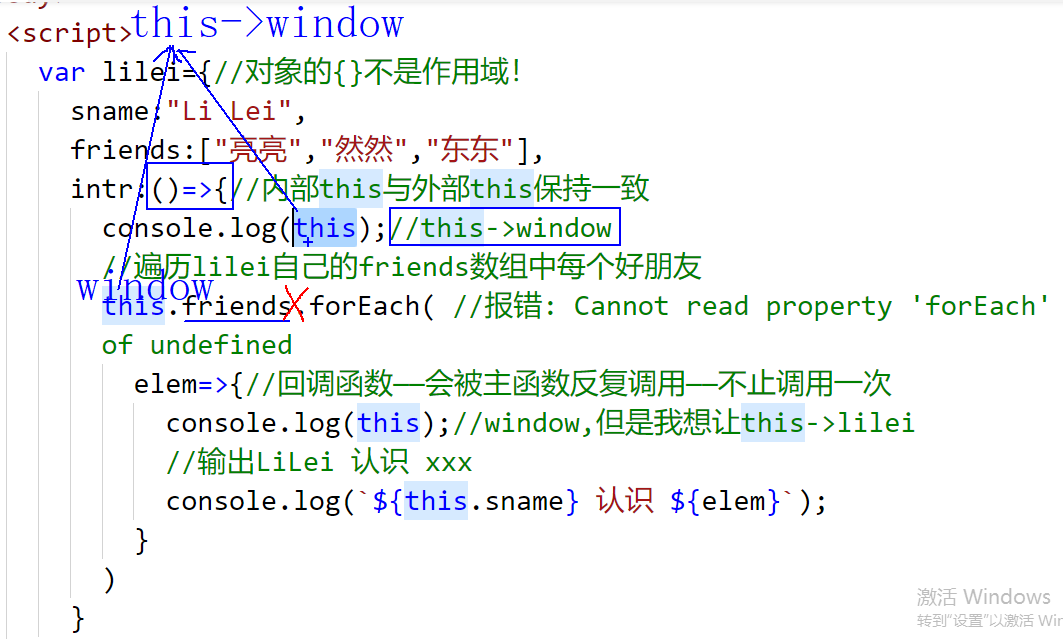
c. 如果函数中用到了this，但是不希望函数中的this与函数外的this保持一致时，则不能用箭头函数简写，典型场景: 对象中的方法不能用箭头函数简写

8. 示例: 使用箭头函数简写回调函数和对象中的方法

10\_arrow\_this.html







|  |
| --- |
| <script>  var lilei={//对象的{}不是作用域！  sname:"Li Lei",  friends:["亮亮","然然","东东"],  intr:function(){  //遍历lilei自己的friends数组中每个好朋友  this.friends.forEach(  elem=>{//回调函数——会被主函数反复调用——不止调用一次  console.log(this);//window,但是我想让this->lilei  //输出LiLei 认识 xxx  console.log(`${this.sname} 认识 ${elem}`);  }  )  }  }  lilei.intr();  function liang(帮别人做一件事){  console.log(`亮哥自己先吃完饭`);  /\* \*/帮别人做一件事();//因为回调函数在主函数内调用时，前边既没有.，又没有new，所以通常回调函数中的this只能默认指window  }  //然哥想让liang帮忙带一屉包子  liang(function(){  console.log(this);  console.log(`帮然哥带包子`)  });  </script>  运行结果：  {sname: "Li Lei", friends: Array(3), intr: ƒ}  Li Lei 认识 亮亮  {sname: "Li Lei", friends: Array(3), intr: ƒ}  Li Lei 认识 然然  {sname: "Li Lei", friends: Array(3), intr: ƒ}  Li Lei 认识 东东  亮哥自己先吃完饭  Window {parent: Window, opener: null, top: Window, length: 0, frames: Window, …}  帮然哥带包子 |

四. for of

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

1. 什么是for of: 对大多数for循环遍历索引数组的简写

2. 何时: 今后绝大多数情况下从头到尾的遍历索引数组和类数组对象，中每个元素都可用for of简化普通for循环

3. 如何:

for(var 变量 of 数组/类数组对象){

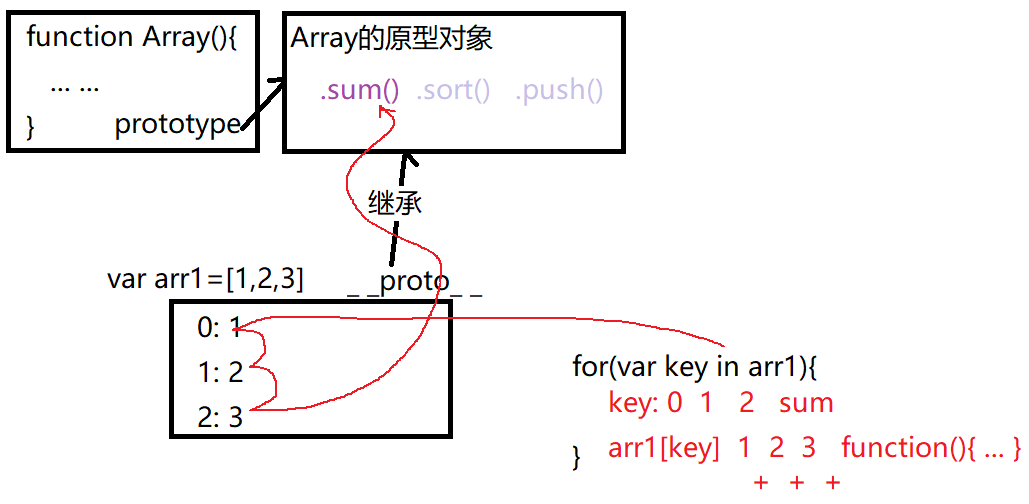
//of会依次取出数组中每个元素值，自动保存到of前的变量中

对of前变量执行操作

}

4. 总结:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | for | arr.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. for of的局限: vs for循环

(1). for(var i=0;i<arr.length;i++)

a. 可改变遍历的顺序: for(var i=arr.length-1;i>=0;i--){ ... }

b. 可改变遍历的步调: for(var i=0;i<arr.length;i+=2)

c. 既可以获得元素的内容arr[i]，又可以获得元素的位置i

(2). for(var elem of arr)

a. 不可改变遍历的顺序，只能从头到尾

b. 不可改变遍历的步调，只能依次挨个遍历

c. 只能获得元素值elem，无法获得元素位置i

6. 总结: 因为绝大多数循环，不需要改变顺序，步调，只关心元素值，不关心位置，所以反而for of用的绝对多！

7. 示例: 使用for of遍历索引数组和类数组对象

11\_for\_of.html

|  |
| --- |
| <script>  var arr=["亮亮","然然","东东"];  //点名  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！`)}  //定义函数对任意个数字求和  function add(){  var sum=0;  //for(var i=0;i<arguments.length;i++){  for(var value of arguments){  sum+=value  }  return sum;  }  console.log(add(1,2,3))//6  console.log(add(1,2,3,4,5))//15  //为数组家添加一个对数组元素求和的共有方法sum  Array.prototype.sum=function(){  //this->将来的arr1, arr2  var total=0;  // for(var i=0;i<this.length;i++){  // total+=this[i]  // }  //this.forEach(elem=>total+=elem);  for(var elem of this){  total+=elem;  }  //错误: in不止遍历当前对象的成员，还会自动沿着原型链向上继续遍历父对象中可用的成员  //in会依次取出in后对象中每个属性名保存到in前的变量key中  // for(var key in this){  // total+=this[key]  // }  return total;  }  var arr1=[1,2,3];  var arr2=[1,2,3,4,5]  console.log(arr1.sum());  console.log(arr2.sum());  </script>  运行结果:  亮亮 - 到！  然然 - 到！  东东 - 到！  亮亮 - 到！  然然 - 到！  东东 - 到！  亮亮 - 到！  然然 - 到！  东东 - 到！  6  15  6  15 |

总结: this 判断this一定不要看定义在哪儿！只看调用时.前是谁或有没有new

4种:

1. obj.fun() this->.前的obj

2. new Fun() this->new正在创建的新对象

3. 构造函数.prototype.共有方法=function(){ ... } this->将来调用这个共有方法的.前的某个子对象

4. fun() 或 (function(){ ... })() 或 回调函数 this->window

ES5: //千万不要背！

1. 严格模式: "use strict"

(1). 禁止给未声明的变量赋值

(2). 静默失败升级为错误

(3). fun() 或 (function(){ ... })() this->undefined

(4). 禁用了arguments.callee

2. 保护对象:

(1). 保护对象属性: 2种:

a. 用开关保护对象的属性:

1). 每个属性有3个开关:

i. writable: true或false 控制是否可修改当前属性值

ii. enumerable: true 或false 控制是否可用for in遍历到该属性

半隐藏(浅紫色)，只防for in 不防.

iii. configurable: true或false 控制2件事:

①是否可删除该属性；②是否可修改前两个特性

一旦改为false 不可逆

2). 修改开关: 2种:

i. 只修改一个属性中的开关:

Object.defineProperty(对象,"属性名",{

开关: true或false,

configurable:false //双保险

})

ii. 同时修改多个属性中的开关:

Object.defineProperties(对象,{

属性名:{

开关: true或false,

configurable:false //双保险

},

属性名:{

开关: true或false,

configurable:false //双保险

},

... ...

})

b. 用自定义规则保护对象的属性: 访问器属性 3步:

Object.defineProperties(对象,{

1). 定义一个隐形埋名且半隐藏的普通属性实际保存属性值

\_属性名:{

value:属性值,

writable: true, //通常受保护的属性都可以修改，只不过满足条件才能改

enumerable:false, //半隐藏，不想被外人轻易发现

configurable:false //以上设置禁止随意修改

},

2). 请保镖: 冒名顶替原属性，一请就是一对儿

对外公开的属性名:{

get:function(){ //负责从受保护的\_属性中获取属性值

return this.\_属性名

},

set:function(value){ //负责将新属性值经过验证后更新到受保护的\_属性中

if(value符合要求){

this.\_属性名=value

}else{

throw Error("错误提示")

}

},

enumerable:true, //保镖要代替被保护的\_属性名抛头露面

configurable:false, //以上配置禁止随意修改

}

})

3). 使用访问器属性: 和使用普通属性是完全一样的。其它程序不知道自己用的是访问器属性——蒙在鼓里。

i. 想获取访问器属性值: 对象.访问器属性

程序会自动调用访问器属性内部的get()，返回受保护的\_属性名

ii. 想修改访问器属性值: 对象.访问器属性=新值

程序会自动调用访问器属性内部的set()，将新值自动传给value形参，如果value形参验证通过，才将新属性值value，保存到受保护的\_属性名中

(2). 保护对象的结构: 3个级别:

a. 防扩展: Object.preventExtensions(对象)

b. 密封: Object.seal(对象)

c. 冻结: Object.freeze(对象)

3. Object.create()

(1). 何时: 即使没有构造函数，只有一个父对象，也想创建子对象继承父对象时都用Object.create()

(2). 如何:

var 子对象名=Object.create(父对象,{

自有属性名:{

value:属性值,

writable: true 或false,

enumerable:true 或 false,

},

自有属性名:{

value:属性值,

writable: true 或false,

enumerable:true 或 false,

},

... ...

})

Object.seal(子对象);

4. 替换this: 3种情况:

(1). 只在一次调用函数时，临时替换一次函数中的this: call 或 apply

a. 多数情况下首先call()

b. 只有在多个实参值被放在一个数组中给定时，需要先打散数组，再传参：apply()

(2). 创建一个新函数并永久绑定this为一个指定的对象: bind()

5. 数组新增函数:

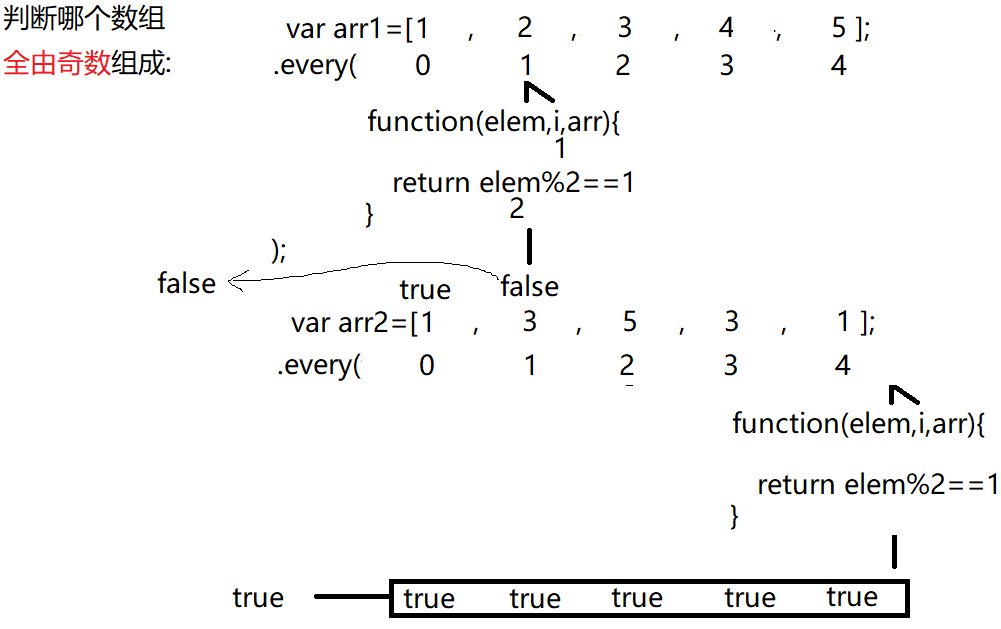
(1). 判断:

a. 判断数组中是否所有元素都符合要求:

arr.every(function(elem,i,arr){

return 判断条件

})

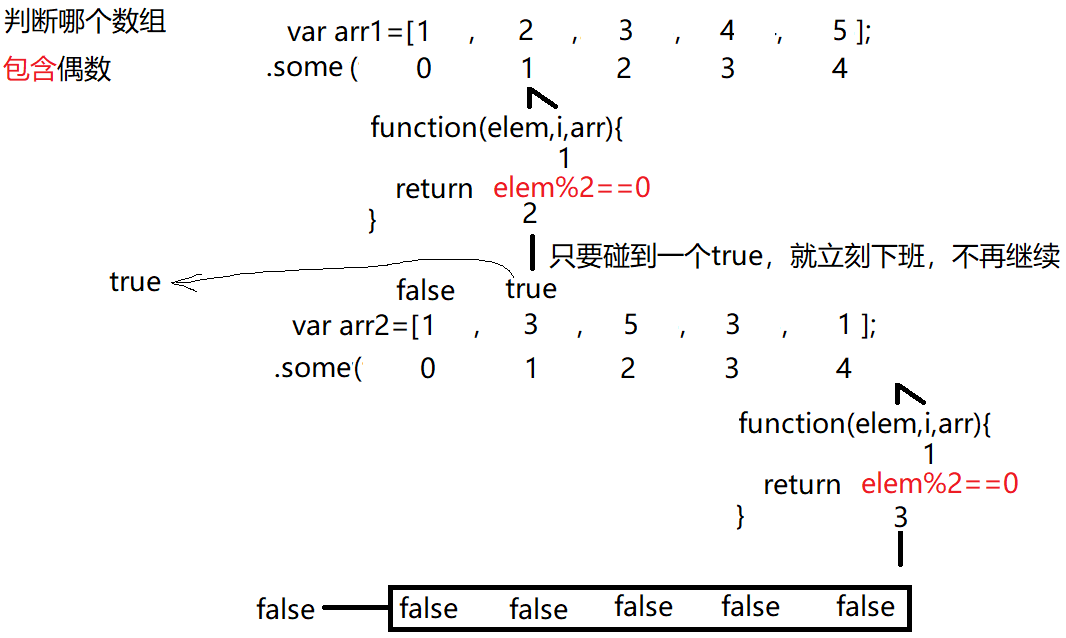


b. 判断数组中是否包含符合要求的元素:

arr.some(function(elem,i,arr){

return 判断条件

})



(2). 遍历: 2个:

a. 如果只是单纯遍历原数组中的值，代替普通for循环: forEach

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

对当前元素值执行的操作

})

b. 如果希望保护原数组，经过修改原数组中的元素值后，返回新数组: map

var 新数组=arr.map(function(elem,i,arr){

return 当前元素值经过加工后返回的新值

})

(3). 过滤:

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

return 判断条件

})

(4). 汇总:

var 汇总结果=arr.reduce(function(捐款箱, elem,i,arr){

return 捐款箱+elem

}, 捐款箱中汇总开始时的起始值)

\*\*\*\*ES6\*\*\*\*

1. 今后，拼接字符串都用模板字符串代替+号拼接

2. 今后，凡是声明变量都用let代替var。(不会被声明提前，不会超出代码块{})

3. 今后，几乎所有function都可用=>简写

特例: 只有不希望函数内this与函数外this保持一致时，才不能用箭头函数简写

4. 今后，凡是从头到尾遍历索引数组全部元素时，且只关心元素内容不关心元素位置的遍历都可用for of简写普通for循环

今日对应小程序视频:

小程序->在线->JSCORE->day06 4. forEach

6. map原理 数组元素\*2...

7. filter

8. reduce原理 简单 数组元素求和

小程序->在线->JSCORE->day07 1. 模板字符串

2. let

3. 箭头函数

4. for of

作业:

1. 复习今日问题清单(小程序->首页->JSCORE->day06、day07)，预习明日问题清单(小程序->首页->JSCORE->day08)

2. 看小程序视频学习ES6中关于对象和属性的简写:

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. 看小程序视频预习参数增强

小程序->在线->JSCORE->day07 5.参数增强

4. 看小程序视频预习解构

小程序->在线->JSCORE->day07 6.解构(上)...

5. 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

6. （学有余力）看小程序视频学习let+for相关笔试题:

小程序->在线->JSCORE->day07 作业: 高频笔试题 定时器setTimeout let 5个5