小程序: WEB问题速查

1. 问题清单: 历届师兄师姐在今天课程中集中问过的高频的重点的问题！

那些自己发现不了问题，又迫切想知道其它同学都问了什么问题，的同学

小程序->首页->JSCORE->day06、day07

当老师在回答其它同学的问题时，你不要闲着！你可以打开问题清单浏览别人曾经问过的经典问题！

2. 高清无码小视频: 小程序->在线->JSCORE->day06、day07

2部分: 1. 课上重点知识和案例的回放！2. 重点高频笔试面试题讲解扩展视频

万一家里或中心网络断了！暂时看不了直播了！

因为某些事儿请假了，错过了当天的直播！

课下复习时，觉得TMOOC视频太大，浏览不方便或TMOOC不能看了！

——打开小程序中"在线"，就可下载观看当前所讲内容和当天所讲全部内容的回放。

基础比较好的，确实学有余力的同学，想拿高薪的同学，在我答疑时，不要闲着:

要看今天扩展视频: 小程序->在线->JSCORE->day06

5. forEach按值传递byValue

作业1: 将类数组对象转成数组...

作业2: 高频笔试题 用原生call模拟实现bind()

作业: 使用reduce统计一个字符串中每种字符出现的次数

作业: 使用原生js仿写forEach和map

3. 单词列表: 小程序->单词->JSCORE->day06、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. 凡是拼写错误，自己懒得找！我可以帮你找，但是找到后，错误的单词你要手抄20遍，并拍照！

出错: 先开f12，console，看错误提示！

3. 每个带注释的案例至少写三遍:

(1). 注释+代码抄一遍

(2). 保留注释，删除代码，自己试着把代码翻译回来！

(3). 删除注释和代码，自己试着用自己的话把注释写回来！

一个例子只有自己可以把人话的注释写回来！才算自己会！

问题: 一听就会一做就废！原因: 你自己无法用人话精确描述自己想干什么！

如何提问: （vip待遇的剩余时间14天）

1. 上课时，案例中，笔记中那句话哪行代码看不懂，立刻问！

2. 小程序中问题清单里哪个问题和回答看不懂！立刻问！

3. 个人项目中哪个功能不会做！立刻问！

4. 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. Object.create()

2. 替换this

3. 数组新函数

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

一. Object.create():

对应小程序视频: 因为太简单，所以，没有视频

1. 问题: 如果在没有构造函数的情况下，也想创建一个子对象，继承父对象

2. 解决: Object.create()

3. 如何: 一句话三件事:

**var 子对象=Object.create(父对象,{**

//语法和defineProperties相同

**属性名:{**

**开关:true或false,**

**... : ...**

**},**

... : {

...

}

})

4. 原理:

1. 先创建一个新的空对象

2. 自动让新对象继承指定的父对象

3. 为子对象添加自有属性

5. 示例: 使用Object.create()创建子对象，继承父对象

1\_Object\_create.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 father={        bal:100000000000,        car:"infiniti"      };      //创建一个新对象lilei，并让李磊自动继承father对象      var lilei=Object.create(father,{        //为李磊添加自有属性sname和sage        sname:{          value:"Li Lei",          writable:true,          enumerable:true        },        sage:{          value:11,          writable:true,          enumerable:true        }      });      //密封李磊对象      Object.seal(lilei);      console.log(lilei);    </script>  </body>  </html>  运行结果:   1. *{sname: "Li Lei", sage: 11}*    1. sage: 11    2. sname: "Li Lei"    3. \_\_proto\_\_:       1. bal: 100000000000       2. car: "infiniti"       3. \_\_proto\_\_: Object |

二. 替换this

对应小程序视频: 小程序->在线->JSCORE->day06 2. 替换this, call,apply,bing

1. 问题: 有时函数中的this指向的对象，不是我们想要的！

2. 解决: 其实如果函数中的this指向的对象不是我们想要的，我们是可以修改的！

3. 第一种: 只在本次调用函数时，**临时**修改一次this指向的对象

(1). **要调用的函数.call(替换this的对象, 实参值, ...)**

(2). 2件事:

a. 调用一次函数执行，同时传入实参值

b. 同时**临时替换**函数中的this为()中第一个值指定的对象

(3). 示例: 定义公共计算器函数，用不同对象替换函数中的this

2\_call.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>      //有一个公共的计算薪资的函数      //              底薪  奖金1  奖金2      function jisuan(base,bonus1,bonus2){        console.log(`${this.ename} 的总工资是: ${base+bonus1+bonus2}`)      }      //有两个员工:      var lilei={ ename:"Li Lei" };      var hmm={ ename:"Han Meimei" };      //两个员工都想用jisuan()函数，计算自己的薪资，都希望计算结果中显示自己的名字      //比如: lilei想调用jisuan()函数，计算自己的薪资，希望计算结果中显示自己的名字"Li Lei"      //其实就是希望把jisuan()函数中的this->lilei      //错误做法1:  /\* \*///jisuan(10000, 1000, 2000)//this->window      //错误做法2:      //lilei.jisuan(10000,1000,2000)//报错: lilei.jisuan is not a function      //一个对象的原型链上保存着这个对象可用的所有成员      //李磊的原型链:      //lilei自己内部没有jisuan()      //lilei的爹也没有jisuan()      //console.log(lilei.\_\_proto\_\_);      //lilei的爷爷是null了，所以更没有jisuan()      //console.log(lilei.\_\_proto\_\_.\_\_proto\_\_);      //结论: jisuan()函数虽然在全局，但是却没有在lilei的原型链上！所以lilei.jisuan()找不到，就会报错！      //正确: //\*临时\*替换this为lilei      jisuan.call( lilei,10000, 1000,  2000);  //                 |     ↓      ↓      ↓  //function jisuan( ↓    base, bonus1, bonus2){    //              this.ename...      //为什么是临时的:      jisuan(10000,1000,2000);//this->window      //hmm也想调用jisuan()函数，计算自己的薪资      jisuan.call(  hmm,  4000, 5000,  6000);  //                 |     ↓      ↓      ↓  //function jisuan( ↓    base, bonus1, bonus2){    //              this.ename...    </script>  </body>  </html>  运行结果:  Li Lei 的总工资是: 13000  undefined 的总工资是: 13000  Han Meimei 的总工资是: 15000 |

4. 第二种情况: 只在本次调用函数时，**临时**修改一次this指向的对象，并打散数组传参

(1). 问题: 有时，多个实参值是放在一个**数组**中整体给我们的！而函数需要的却是分散的多个实参值！出现了不一致

(2). 解决: 用apply()代替call()

(3). 如何: **要调用的函数.apply(替换this的对象, 数组)**

(4). 原理:

a. 调用一次函数执行，同时先打散数组参数为多个实参值，再传入

b. 同时临时替换函数中的this为()中第一个实参值指定的对象

(5). 示例: 如果李磊的薪资是放在一个数组中整体给的，如何替换this并传参？

3\_apply.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>      //有一个公共的计算薪资的函数      //              底薪  奖金1  奖金2      function jisuan(base,bonus1,bonus2){        console.log(`${this.ename} 的总工资是: ${base+bonus1+bonus2}`)      }      //有两个员工:      var lilei={ ename:"Li Lei" };      var hmm={ ename:"Han Meimei" };      //李磊的三个薪资是放在一个数组中给的      var arr=[10000,1000,2000];      //错误做法:      //jisuan.call(lilei,arr/\*,undefined,undefined\*/)      //             |     ↓      ↓           ↓  //function jisuan( ↓    base,  bonus1,   bonus2)      //           this.ename      //         [10000,1000,2000]+undefined+undefined      //自动将数组转为字符串，拼接字符串——不是我们想要的！      //正确做法: 先打散数组为多个值，再传参      jisuan.apply(lilei,          arr               )  //                 |           打散数组  //                 |    10000，  1000，  2000  //                 |      ↓       ↓        ↓  //function jisuan( ↓    base,  bonus1,   bonus2)      //           this.ename    </script>  </body>  </html>  运行结果:  Li Lei 的总工资是: 13000 |

5. 第三种情况: 基于原函数，创建一个新函数副本，并永久绑定this为指定对象

(1). 问题: 如果一个函数需要反复调用，但是每次都被迫用call()或apply()替换this，极其不方便！

(2). 解决: 完全可以"买"一个一模一样的新函数副本，在新函数副本中**永久**绑定想用的this！

(3). 如何: **var 新函数名=原函数.bind(替换this的对象)**

(4). 原理:

a. 创建一个和原函数一模一样的新函数！~~不是调用函数！~~

b. 永久的替换新函数中this为指定的对象

(5). 问题: 如果每次调用函数时，个别实参值也是固定的！每次都需要传入固定的个别实参值——很麻烦的！

(6). 解决: 其实bind不但可以永久替换this，而且还能**永久绑定个别实参值**

**var 新函数名=原函数.bind(替换this的对象, 固定实参值, ...)**

(7). 原理:

a. 创建一个和原函数一模一样的新函数！~~不是调用函数！~~

b. 永久的替换新函数中this为指定的对象

c. 永久替换对应位置的形参变量为一个固定的值！

(8). 示例: 创建一个新的计算器函数，永久绑定this为lilei，base为10000

4\_bind.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>      //有一个公共的计算薪资的函数      //              底薪  奖金1  奖金2      function jisuan(base,bonus1,bonus2){        console.log(`${this.ename} 的总工资是: ${base+bonus1+bonus2}`)      }      //有两个员工:      var lilei={ ename:"Li Lei" };      var hmm={ ename:"Han Meimei" };      //李磊觉得总是跟别人共用一个，总是借用很不方便！      //且李磊的底薪每个月都是固定的！不想每次都输入10000      // jisuan.call(lilei,10000,1000,2000);      // jisuan.call(lilei,10000,2000,2000);      // jisuan.call(lilei,10000,1000,1500);      //于是lilei决定买一个新的一模一样的，自己专属的计算器      var jisuan2=jisuan.bind(lilei,10000);  //                            |    ↓永久  //         jisuan2:function(  |   base,...){                              //↓   base=10000                             //this->lilei    // console.log(`${lilei.ename} 的总工资是: ${base+bonus1+bonus2}`)  //}      //从此李磊使用计算2时，再不用call了！再不用借了！      //且每个月计算薪资时不用重复输入底薪了      jisuan2(1000,2000);      jisuan2(2000,2000);      jisuan2(2000,1500);      //结论: 每次调用时即使不传入lilei，也能输出lilei说明是永久的替换this！即使不传入实参值10000，也能将10000累加到总工资，说明base也永久绑定为10000了！    </script>  </body>  </html>  运行结果:  Li Lei 的总工资是: 13000  Li Lei 的总工资是: 14000  Li Lei 的总工资是: 13500 |

6. 总结: 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, ...)

三. 数组新函数:

1. 判断:

对应小程序视频: 小程序->在线->JSCORE->day06 3. every原理...

(1). every: 判断数组中是否所有元素都符合要求

a. **var 判断结果=arr.every(function(元素值, 下标, 当前数组){**

**return 根据元素值，下标和当前数组，判断当前元素是否符合要求**

**})**

说明: 如果回调函数中的判断条件用不到后两个实参值，可以省略后两个形参。

b. 原理:

1). every中自带for循环，自动遍历.前的数组中每个元素

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，都自动传入三个值:

i. 第一个实参值value: 自动收到当前正在遍历的元素值

ii. 第二个实参值i: 自动收到当前正在遍历的元素位置

iii. 第三个实参值arr: 自动收到当前every.前的数组对象

4). 回调函数中用传入的三个值验证当前正在遍历的元素是否符合条件的要求！并将判断结果返回给every()函数

5). 2种情况:

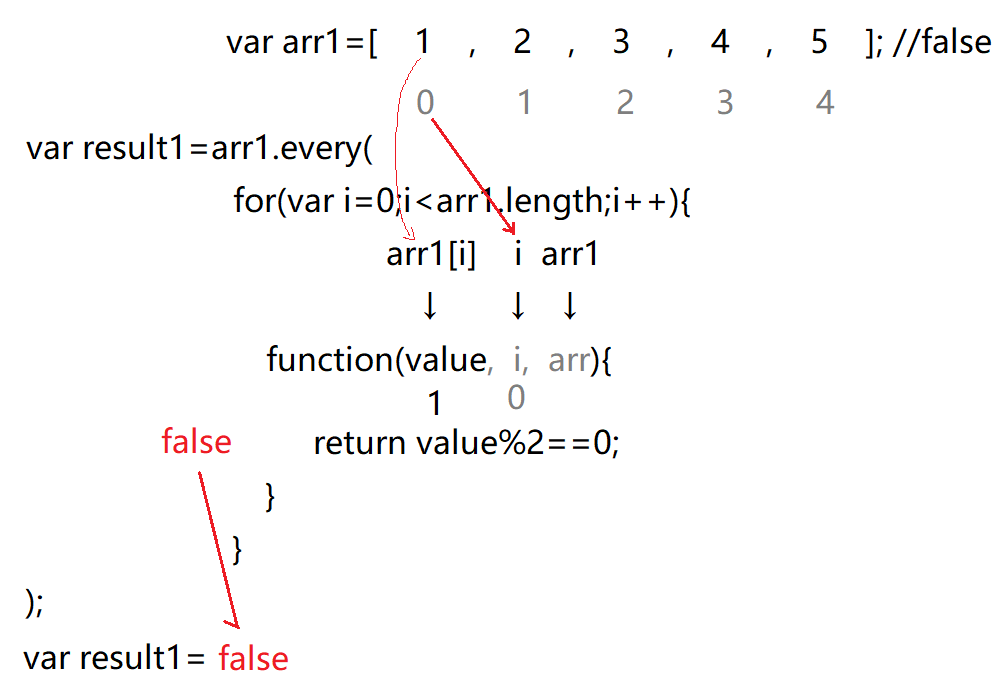
i. 如果本次回调函数返回的true，说明当前元素符合要求！于是every会继续向后遍历，直到遍历结束。如果遍历结束，所有元素的验证结果都返回true，则整个every就返回true，说明整数数组中所有元素都符合要求

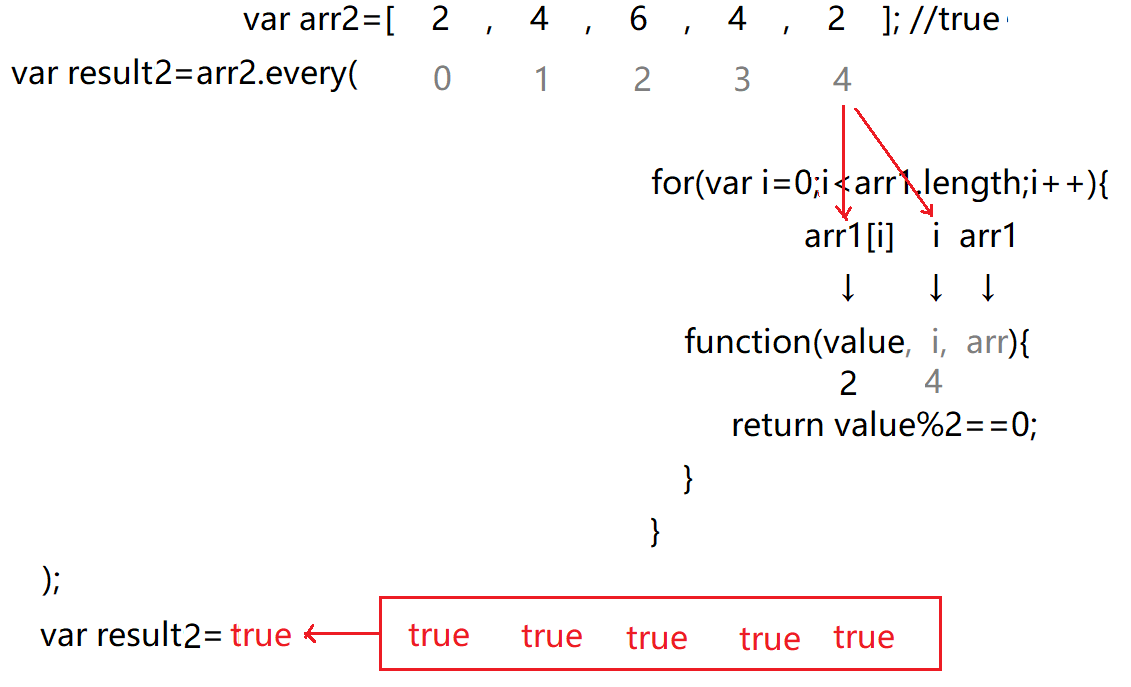
ii. 如果本次回调函数返回false！说明当前元素不符合要求！every就没必要继续循环！而是立刻退出执行。整个every()返回false，说明数组不是所有元素都符合要求！

c. 示例: 使用every函数，判断哪个数组全由偶数组成:

5\_every.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=[1,2,3,4,5]; //false      var arr2=[2,4,6,4,2]; //true      //判断哪个数组全由偶数组成:      //                              三个变量名都能换      //                              但是顺序不能变      //                          当前元素值 下标 数组      //                                ↓     ↓   ↓  **var result1=arr1.every(function(value,  i, arr){**        console.log(`arr1.every()自动调用了一次回调函数；自动传入arr1数组:[${arr}]的第${i}个元素值:${value}。经过回调函数验证，返回${value%2==0}`)        //每个元素是否都是偶数        return value%2==0;//当前元素值是否为偶数      });      var result2=arr2.every(function(value, i , arr){        console.log(`arr2.every()自动调用了一次回调函数；自动传入arr2数组:[${arr}]的第${i}个元素值:${value}。经过回调函数验证，返回${value%2==0}`)        return value%2==0;//当前元素值是否为偶数      });      console.log(result1, result2);    </script>  </body>  </html>  运行结果：  arr1.every()自动调用了一次回调函数；自动传入arr1数组:[1,2,3,4,5]的第0个元素值:1。经过回调函数验证，返回false  arr2.every()自动调用了一次回调函数；自动传入arr2数组:[2,4,6,4,2]的第0个元素值:2。经过回调函数验证，返回true  arr2.every()自动调用了一次回调函数；自动传入arr2数组:[2,4,6,4,2]的第1个元素值:4。经过回调函数验证，返回true  arr2.every()自动调用了一次回调函数；自动传入arr2数组:[2,4,6,4,2]的第2个元素值:6。经过回调函数验证，返回true  arr2.every()自动调用了一次回调函数；自动传入arr2数组:[2,4,6,4,2]的第3个元素值:4。经过回调函数验证，返回true  arr2.every()自动调用了一次回调函数；自动传入arr2数组:[2,4,6,4,2]的第4个元素值:2。经过回调函数验证，返回true  false true |





(2). some: 判断数组中是否包含符合要求的元素

a. **var 判断结果=arr.some(function(元素值, 下标, 当前数组){**

return 根据元素值，下标和当前数组，判断当前元素是否符合要求

})

b. 原理:

1). some中自带for循环，自动遍历.前的数组中每个元素

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，都自动传入三个值:

i. 第一个实参值value: 自动收到当前正在遍历的元素值

ii. 第二个实参值i: 自动收到当前正在遍历的元素位置

iii. 第三个实参值arr: 自动收到当前some.前的数组对象

4). 回调函数中用传入的三个值验证当前正在遍历的元素是否符合条件的要求！并将判断结果返回给some()函数

5). 2种情况:

i. 如果回调函数当前元素的验证结果返回true，说明当前元素符合要求，则some()立刻退出循环，不再继续执行。而且整个some()返回true，说明当前数组包含至少一个符合要求的元素。

ii. 如果回调函数当前元素的验证结果返回false，说明当前元素不符合要求，则some()会继续向后执行！如果所有元素的验证结果都为false，说明数组中并不包含任何一个符合要求的元素。则整个some()返回false！

c. 示例: 使用some函数，判断哪个数组包含偶数:

6\_some.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=[1,2,3,4,5]; //true 2次      var arr2=[2,4,6,4,2]; //true 1次      //判断哪个数组包含偶数:      //                              三个变量名都能换      //                              但是顺序不能变      //                          当前元素值 下标 数组      //                                ↓     ↓   ↓  **var result1=arr1.some(function(value,  i, arr){**        console.log(`arr1.every()自动调用了一次回调函数；自动传入arr1数组:[${arr}]的第${i}个元素值:${value}。经过回调函数验证，返回${value%2==0}`)        //每个元素是否都是偶数        return value%2==0;//当前元素值是否为偶数      });      var result2=arr2.some(function(value, i , arr){        console.log(`arr2.every()自动调用了一次回调函数；自动传入arr2数组:[${arr}]的第${i}个元素值:${value}。经过回调函数验证，返回${value%2==0}`)        return value%2==0;//当前元素值是否为偶数      });      console.log(result1, result2);    </script>  </body>  </html>  运行结果：  arr1.every()自动调用了一次回调函数；自动传入arr1数组:[1,2,3,4,5]的第0个元素值:1。经过回调函数验证，返回false  arr1.every()自动调用了一次回调函数；自动传入arr1数组:[1,2,3,4,5]的第1个元素值:2。经过回调函数验证，返回true  arr2.every()自动调用了一次回调函数；自动传入arr2数组:[2,4,6,4,2]的第0个元素值:2。经过回调函数验证，返回true  true true |

2. 遍历:

(1). forEach: 单纯简化for循环！

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

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

**对value，i或arr执行操作**

**})**

b. 原理:

1). forEach中自带for循环，自动遍历.前的数组中每个元素

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，都自动传入三个值:

i. 第一个实参值value: 自动收到当前正在遍历的元素值

ii. 第二个实参值i: 自动收到当前正在遍历的元素位置

iii. 第三个实参值arr: 自动收到当前forEach.前的数组对象

4). 回调函数中对当前元素值，数组中当前位置的元素执行相同的操作，不需要返回值

c. 为什么: for循环已经没有进一步简化的空间了！且每次都写for循环，太麻烦了！

d. 问题: forEach是数组家的函数，只有遍历索引数组时，才能使用forEach。如果遍历类数组对象，虽然也是数字下标，但是不能使用forEach。

e. 示例: 使用forEach简化遍历数组中每个元素

7\_forEach.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"];      //遍历花名册中每个人名，点到谁，谁喊到！      //for循环      for(var i=0;i<arr.length;i++){        alert(`${arr[i]} - 到！`)      }      //forEach      // arr.forEach(function(value){      //   alert(`${value} - 到!`)      // })      //forEach+箭头函数(暂时不要问！后边详细讲)      //arr.forEach((value)=>{alert(`${value} - 到!`)})      //箭头函数，如果只有一个形参，可省略()      //arr.forEach(value=>{alert(`${value} - 到!`)})      //箭头函数，如果只有一句函数体，可省略{}      //arr.forEach(value=>alert(`${value} - 到!`))      //不愿意写那么长单词      arr.forEach(v=>alert(`${v} - 到!`))    </script>  </body>  </html>  运行结果: |

(2). map: 遍历出原数组中每个元素值，加工后，放入新数组中返回！

对应小程序视频: 小程序->在线->JSCORE->day06 6. map...

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

**return 加工后的新元素值**

**})**

b. 原理:

1). arr.map()先创建一个空数组等待

2). map()也自带for循环自动遍历原数组中每个元素

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，都自动传入三个值:

i. 第一个实参值value: 自动收到当前正在遍历的元素值

ii. 第二个实参值i: 自动收到当前正在遍历的元素位置

iii. 第三个实参值arr: 自动收到当前map.前的数组对象

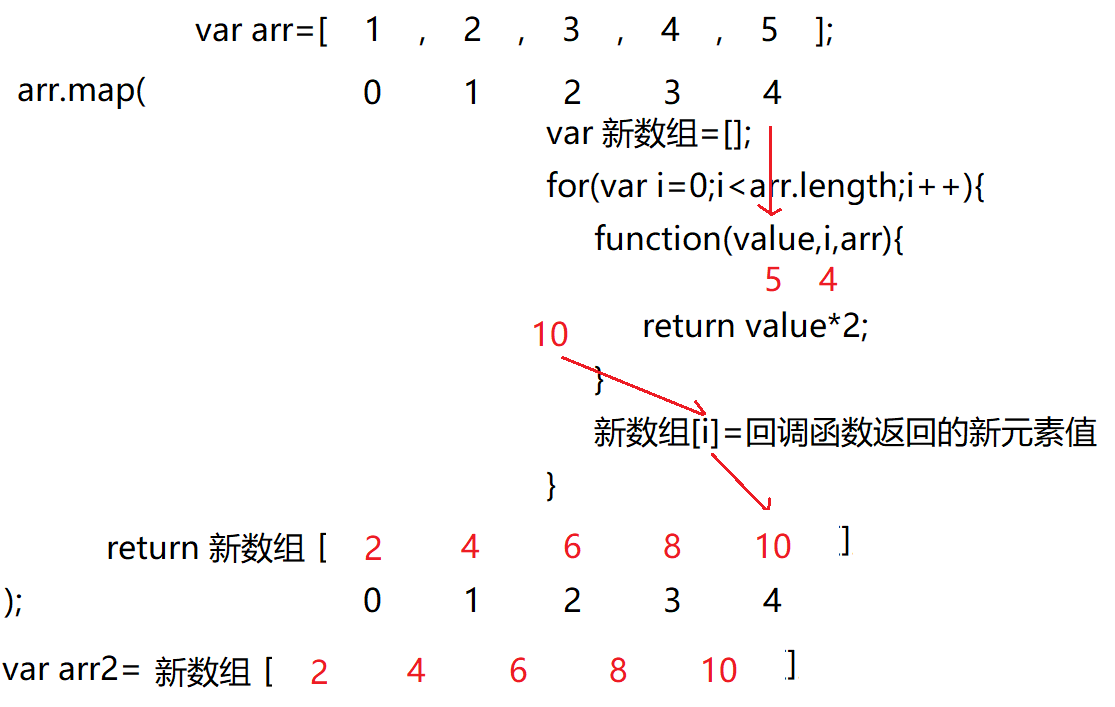
5). 回调函数中根据当前传入的元素值等信息，加工出一个新的元素值返回给map()

6). map()会将收到的新元素值放入新数组中相同位置保存起来

7). 遍历结束，map()会将新数组返回出来！——原数组保持不变

c. 何时: 今后只要希望保护原数组不变，返回新数组供其它位置的代码使用时

d. 示例: 对数组中每个值\*2，返回新数组，原数组保持不变



8\_map.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=[1,2,3,4,5];      //想对数组中每元素\*2，返回新数组，但是，想保持原数组不变！      var arr2=arr.map(function(value,i,arr){        console.log(`arr.map()自动调用了一次回调函数。将原数组[${arr}]中${i}位置的旧值${value}，传入回调函数。经过回调函数加工，返回新元素值${value\*2},自动放入新数组${i}位置`);        //想把当前元素值\*2后的新值放入新数组中？        return value\*2;      });      console.log(arr2);      console.log(arr);    </script>  </body>  </html>  运行结果：  arr.map()自动调用了一次回调函数。将原数组[1,2,3,4,5]中0位置的旧值1，传入回调函数。经过回调函数加工，返回新元素值2,自动放入新数组0位置  arr.map()自动调用了一次回调函数。将原数组[1,2,3,4,5]中1位置的旧值2，传入回调函数。经过回调函数加工，返回新元素值4,自动放入新数组1位置  arr.map()自动调用了一次回调函数。将原数组[1,2,3,4,5]中2位置的旧值3，传入回调函数。经过回调函数加工，返回新元素值6,自动放入新数组2位置  arr.map()自动调用了一次回调函数。将原数组[1,2,3,4,5]中3位置的旧值4，传入回调函数。经过回调函数加工，返回新元素值8,自动放入新数组3位置  arr.map()自动调用了一次回调函数。将原数组[1,2,3,4,5]中4位置的旧值5，传入回调函数。经过回调函数加工，返回新元素值10,自动放入新数组4位置  (5) [2, 4, 6, 8, 10]  (5) [1, 2, 3, 4, 5] |

3. filter过滤: **复制**出原数组中符合要求的元素值放入新数组中返回，原数组保持不变！

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

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

**return 判断条件**

**})**

(2). 原理:

a. filter()先创建一个空数组等待

b. filter ()也自带for循环自动遍历原数组中每个元素

c. 每遍历到一个元素，就自动调用一次回调函数

d. 每次调用回调函数时，都自动传入三个值:

1). 第一个实参值value: 自动收到当前正在遍历的元素值

2). 第二个实参值i: 自动收到当前正在遍历的元素位置

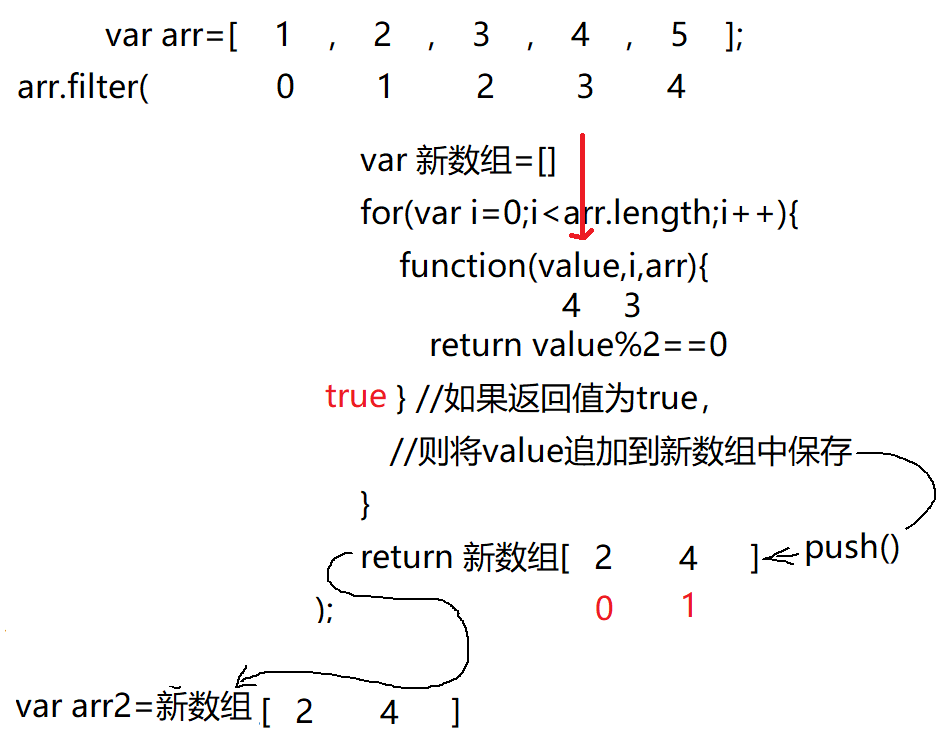
3). 第三个实参值arr: 自动收到当前filter.前的数组对象

e. 在回调函数中判断当前传入的元素值是否符合要求，并返回判断结果

f. 如果本次函数调用结果返回true，说明当前元素符合要求，则filter()自动将当前元素追加到新数组中保存。如果本次函数调用结果返回false，说明当前元素不符合要求，则filter()本次什么也不做，而是继续遍历下一个元素。

g. 遍历结束，filter()将新数组返回

(3). 示例: 过滤出数组中偶数位置的元素，放入新数组中返回



9\_filter.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=[1,2,3,4,5];      //希望过滤出数组中的偶数，放入新数组返回  **var arr2=arr.filter(function(value,i,arr){**        console.log(`arr.filter自动调用了一次回调函数。将原数组${arr}中${i}位置的元素值${value}传入回调函数。经过判断返回${value%2==0}。`)        //判断当前元素是否为偶数  **return value%2==0**      });      console.log(arr2);      console.log(arr);    </script>  </body>  </html>  运行结果:  arr.filter自动调用了一次回调函数。将原数组1,2,3,4,5中0位置的元素值1传入回调函数。经过判断返回false。  arr.filter自动调用了一次回调函数。将原数组1,2,3,4,5中1位置的元素值2传入回调函数。经过判断返回true。  arr.filter自动调用了一次回调函数。将原数组1,2,3,4,5中2位置的元素值3传入回调函数。经过判断返回false。  arr.filter自动调用了一次回调函数。将原数组1,2,3,4,5中3位置的元素值4传入回调函数。经过判断返回true。  arr.filter自动调用了一次回调函数。将原数组1,2,3,4,5中4位置的元素值5传入回调函数。经过判断返回false。  (2) [2, 4]  (5) [1, 2, 3, 4, 5] |

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

对应小程序视频: 小程序->在线->JSCORE->day06 8. reduce...

(1). **var 结果=arr.reduce(**

**function(临时汇总值, 当前元素值, 下标, 当前数组){**

**return 临时汇总值和当前元素值计算出的新临时汇总值**

**} ,**

**起始值**

**)**

(2). 原理:

a. reduce()先用一个变量保存住起始值，遍历过程中，也可保存临时汇总值。

b. reduce()自带for循环，自动遍历原数组中每个元素

c. 每遍历一个元素就自动调用一次回调函数

d. 每次调用回调函数时，都传入保存四个值:

1). 第一个实参值**box:** 自动传入截止到目前的临时汇总值

2). 第二个实参值value: 自动收到当前正在遍历的元素值

3). 第三个实参值i: 自动收到当前正在遍历的元素位置

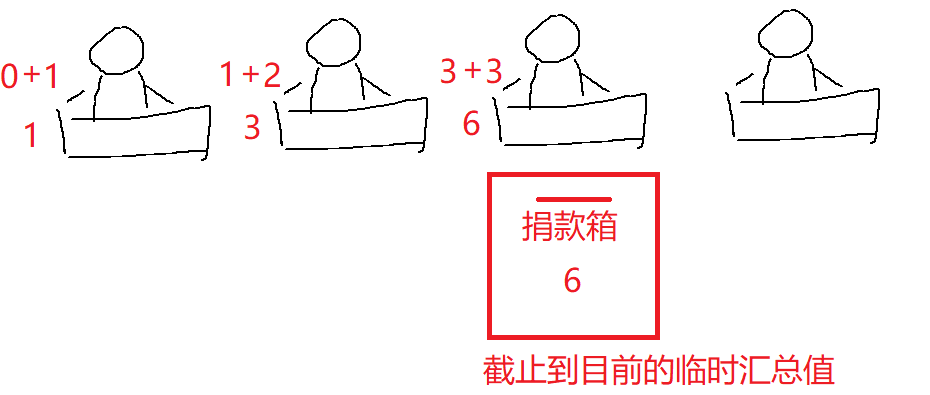
4). 第四个实参值arr: 自动收到当前filter.前的数组对象

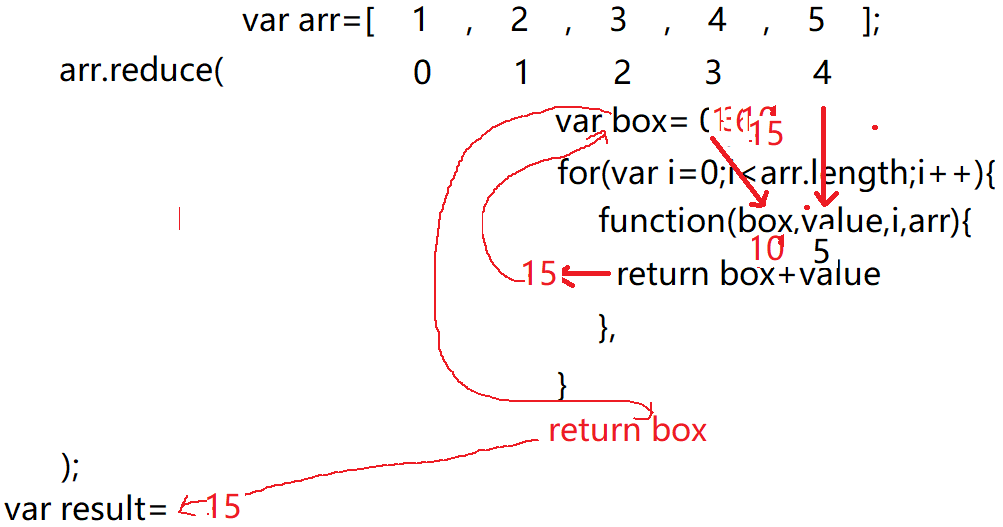
e. 回调函数内，可将当前元素的值，汇总到临时汇总值中，形成新的汇总值，并返回给reduce()

f. reduce()接到回调函数返回的新临时汇总值之后，会覆盖之前旧的临时汇总值保存在变量中，为继续汇总下一个元素值做准备！

g. 遍历结束，最后的临时汇总值，就是整个数组中所有元素的最终汇总值。

(3). 示例: 计算数组中所有元素的和





10\_reduce.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=[1,2,3,4,5];      //想对数组内容求和      var result=arr.reduce(        function(box,value,i,arr){          console.log(`arr.reduce()自动调用了一次回调函数。传入截止目前的临时汇总值box=${box}，和当前元素值${value}，返回box+value的新汇总值${box+value}`)          return box+value        },  **0 //从0开始累加**      );      console.log(result);    </script>  </body>  </html>  运行结果:  arr.reduce()自动调用了一次回调函数。传入截止目前的临时汇总值box=0，和当前元素值1，返回box+value的新汇总值1  arr.reduce()自动调用了一次回调函数。传入截止目前的临时汇总值box=1，和当前元素值2，返回box+value的新汇总值3  arr.reduce()自动调用了一次回调函数。传入截止目前的临时汇总值box=3，和当前元素值3，返回box+value的新汇总值6  arr.reduce()自动调用了一次回调函数。传入截止目前的临时汇总值box=6，和当前元素值4，返回box+value的新汇总值10  arr.reduce()自动调用了一次回调函数。传入截止目前的临时汇总值box=10，和当前元素值5，返回box+value的新汇总值15  15 |

总结:

5. 严格模式: "use strict";

(1). 禁止给未声明过的变量赋值

(2). 静默失败升级为错误

(3). 普通函数调用中的this不指window，而是指undefined

(4). 禁用arguments.callee

6. 保护对象:

(1). 保护属性:

a. 每个属性包含三个开关:

1). writable: 控制是否可修改属性值

2). enumerable: 控制着是否可被for in遍历到，但是只防for in不防.

3). configurable: 控制

i. 是否可删除当前属性

ii. 是否可修改writable和enumerable两个开关

强调: configurable一旦改为 false，不可逆！

b. 只修改一个属性的多个开关:

Object.defineProperty(对象名, "属性名",{开关: true/false})

c. 修改多个属性的多个开关:

Object.defineProperties(对象名,{

属性名:{ 开关:true/false, ... },

... : ...

})

d. 如果用自定义的规则保护属性时，只能用访问器属性: 2步:

Object.defineProperties(对象,{

//1). 先定义一个隐姓埋名且半隐藏的数据属性:

\_属性名:{

value: 属性的初始值,

writable:true,

enumerable:false,

configurable:false

},

//2). 再定义访问器属性保镖冒名顶替要保护的属性

属性名:{

get:function(){

return this.\_属性名

},

set:function(value){ //value ← 要修改的新属性值

先验证value

如果验证通过，this.\_属性名=value

否则如果验证未通过，不但不保存新属性值，还会报错

},

enumerable:true,

configurable:false

}

})

外界试图获取访问器属性值时，自动调用get()

外界试图修改访问器属性值时，自动调用set()

(2). 保护结构: 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)

7. 如果没有构造函数，也想创建子对象，继承父对象:

var 新子对象=Object.create(父对象,{

自有属性:{

value:属性值,

开关:true或false,

... :...

},

... : { ... }

})

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)

|  |
| --- |
| 总结: this 4种: 判断this，~~一定不要看定义在哪儿！~~只看调用时！  1. obj.fun() this->obj  2. fun() 或 (function(){ ... })() this->window  3. new Fun() this->new正在创建的新对象  4. 类型名.prototype.共有方法=function(){ ... } this->将来谁调用指谁，同第一种情况 |

今日小程序视频列表:

对应小程序视频: 小程序->在线->JSCORE->day06 2. 替换this, call,apply,bind

3. every原理...

4. forEach ...

6. map ...

7. filter

8. reduce ...

作业:

1. 复习今日问题清单: 小程序->首页->JSCORE->day06、day07

2. 看小程序视频学习数组家indexOf函数

小程序->在线->JSCORE->day06 作业: 数组家也有indexOf()

3. 看小程序视频学习判断数组是否升序排列:

小程序->在线->JSCORE->day06 3.1 every 判断数组是否升序排列

4. 预习:

小程序->在线->JSCORE->day07 1. 模板字符串

2. let

3. 箭头函数

4. for of

5. 参数增强

6. 解构