提前准备:

1. 今天的历届师兄师姐问过的经典的高频的问题清单都已经整理在小程序->首页->JSCORE->day07里

2. 今天的课程中所需的单词列表都已经整理在小程序->单词->JSCORE->day07里

3. 今天的重点视频、扩展高频鄙视题视频已经整理在小程序->在线->JSCORE->day07里

如果不知道问什么: 谁问的越多！谁培训费花的越值！

1. 笔记中或画图中那句话或者那个位置不明白，立刻问！

2. 案例中哪行代码或注释不明白，立刻问！

3. 课下自己做项目，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！——公众号: 前端大全 定期总结大厂笔试题

5. 看小程序问题清单时，遇到看不懂的问题，立刻问！

衡量是否学会的标准：

1. 可以用自己的话把注释的步骤写出来！才算会！（函数名或程序忘了都没关系！）

2. 可以自己画出内存变化过程图！才算会！（凡是只用眼睛看想得出揭露的！就都是错误的结论！）

3. 可以把老师上课的案例，经过修改后，应用到自己的项目里

正课:

1. 数组新函数: reduce

/\*ES6\*/

1. 模板字符串
2. let
3. 箭头函数
4. for of
5. 参数增强
6. 解构
7. 数组新函数: reduce
8. 什么是: 专门对数组中元素值进行统计的函数
9. 何时: 今后只要对数组中的元素值进行统计时，都用reduce
10. 如何:

var 汇总值=arr.reduce(

function(prev, elem, i, arr){ //可以有四个形参变量

//因为最终更要返回整个数组的汇总值，所以回调函数必须有return

return 新的汇总值

},

起始值 //虽然可以省略，但是强烈不建议省略！

)

1. 意为:

(1). 先将起始值保存在一个变量”prev”中，备用

(2). reduce()中也自带for循环，自动遍历数组中每个元素值

(3). 每遍历一个元素，就自动调用一次回调函数，每次调用回调函数时都会自动传入四个实参值

a. prev: 保存截止到当前元素之前，临时的汇总值

b. elem: 接收当前正在遍历的元素值

c. i: 接收当前正在遍历的位置

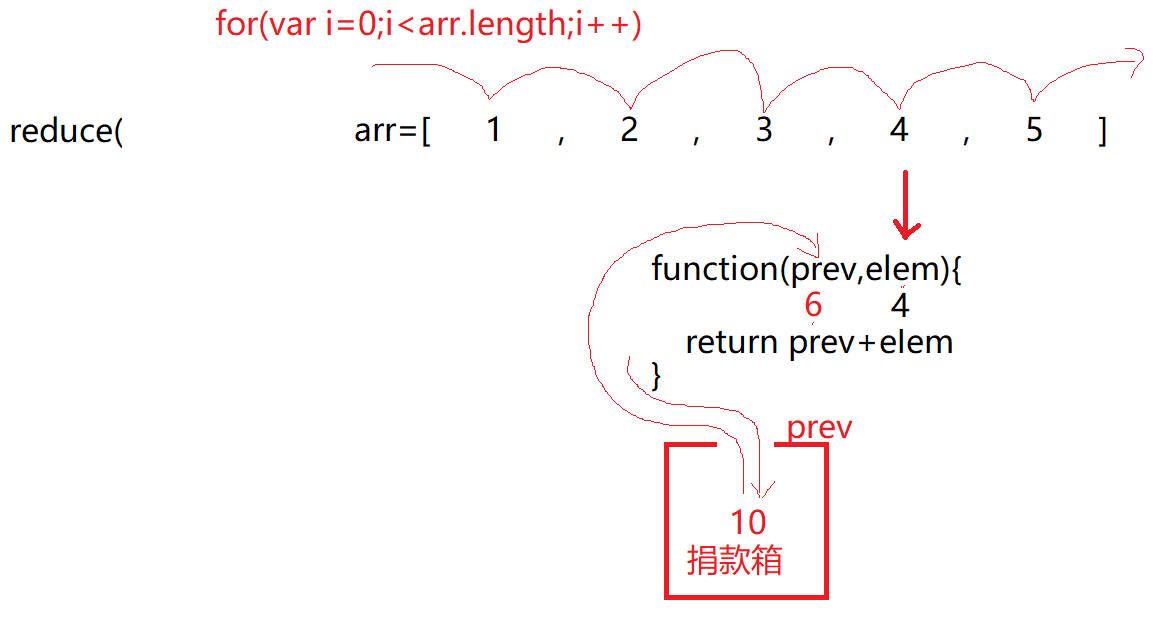
d. arr: 代表.前的数组对象

(4). 回调函数内，应该将本次遍历到的元素，汇总到临时汇总值prev中，并返回新的汇总值prev

(5). reduce()每接到一次回调函数返回的临时汇总值，都会替换变量prev中的旧的汇总值。

(6). 当遍历结束reduce()会将prev变量中保存的最终的汇总值，返回

5. 示例: 对数组中元素值求和：



|  |
| --- |
| var arr=[1,2,3,4,5];  //想对数组中所有元素求和  var result=arr.reduce(    function(prev,elem){      //把当前元素值和截止到目前旧的汇总值相加，计算出新的汇总值，返回给reduce函数      return prev+elem;    },    0 //从0开始累加  );  console.log(result);//15 |

1. 复习模板字符串:
2. 什么是模板字符串: 支持换行和动态生成内容的特殊字符串
3. 为什么: 旧js中用+拼接字符串，极容易和算术计算中的+法计算混淆
4. 何时: 今后只要拼接字符串时，都要用模板字符串，代替+
5. 如何:

(1). 整个字符串要用一对儿反引号` `包裹(键盘上左上角,ESC键正下方的键)

(2). 在``中可以支持换行！

(3). 在``中可以支持动态生成内容。

但是动态生成的内容必须放在${}中

1. ${}中可以放什么: 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. ${}中不能做的: 不能写程序结构(分支和循环)，不能写没有返回值的函数或方法调用！
3. 示例: 尝试模板字符串中拼接各种值

|  |
| --- |
| var price=12.5;  var count=5;  console.log(`    单价:¥${price.toFixed(2)},    数量:${count}  -------------------------------    小计:¥${(price\*count).toFixed(2)}  `);  var orderTime=1584497460171;  console.log(`下单时间:${new Date(orderTime).toLocaleString()}`);  //复习: 思维导图->Date  var sex=1;  console.log(`性别:${sex==1?"男":"女"}`);  var week=["日","一","二","三","四","五","六"];  //         0    1    2    3    4   5    6  console.log(`今天星期${week[new Date().getDay()]}`) |

1. let
2. 什么是: 专门代替var，用来声明变量的新关键词
3. 为什么: 旧js中var有两个广受诟病的缺陷:

(1). 会被声明提前，打乱了程序正常的执行顺序

(2). 没有块级作用域，代码块内的变量，会影响块外的变量

3. 示例: 暴露var的问题:

|  |
| --- |
| var t=0;//全局变量，用来累积每个函数执行的时间  function fun1(){    console.log(`fun1执行耗费0.3s`);    t+=0.3;  }  function fun2(){    console.log(`fun2执行耗费0.8s`);    t+=0.8;    //过了一段时间，老板让我修改一个fun2函数    //实现如果出错，就显示出错的时间    if(false){ //因为条件不满足，所以以下代码没有执行！      var t=new Date();//获得当前系统时间      console.log(`出错啦，at: ${t.toLocaleString()}`)    }  }  fun1();  fun2();  console.log(t);//希望是1.1，但是是0.3，少0.8 |

4. 何时: 今后只要声明变量，都用let，代替var

5. let的优势:

(1). 禁止变量声明提前，保证程序按顺序执行代码！不会破坏程序正常的执行顺序

(2). 让程序结构(分支和循环)的{}，也成为一级作用域——块级作用域，保证块内的变量，不会影响块外的程序。

6. 示例: 用let代替var，解决问题

|  |
| --- |
| var t=0;//全局变量，用来累积每个函数执行的时间  function fun1(){    console.log(`fun1执行耗费0.3s`);    t+=0.3;  }  function fun2(){    console.log(`fun2执行耗费0.8s`);    t+=0.8;    //过了一段时间，老板让我修改一个fun2函数    //实现如果出错，就显示出错的时间    if(true){ //就算这段代码执行，也不会影响if外的程序了      let t=new Date();//获得当前系统时间      console.log(`出错啦，at: ${t.toLocaleString()}`)    }  }  fun1();  fun2();  console.log(t);//1.1 |

7. 原理:

(1). 其实let会被自动翻译为匿名函数自调！

(2). 为了双保险，let的变量还会被悄悄改名: let t=xxx; 可能会被悄悄改为: var \_t=xxx;

8. 实例: 使用匿名函数自调实现等效于let的效果

|  |
| --- |
| var t=0;//全局变量，用来累积每个函数执行的时间  function fun1(){    console.log(`fun1执行耗费0.3s`);    t+=0.3;  }  function fun2(){    console.log(`fun2执行耗费0.8s`);    t+=0.8;    //过了一段时间，老板让我修改一个fun2函数    //实现如果出错，就显示出错的时间    if(true){ //就算这段代码执行，也不会影响if外的程序了      (function(){        var \_t=new Date();        //let t=new Date();//获得当前系统时间        console.log(`出错啦，at: ${\_t.toLocaleString()}`)      })();    }  }  fun1();  fun2();  console.log(t);//1.1 |

9. let的三个小脾气:

(1). 同一个作用域内，禁止再let之前，提前使用该变量

(2). 即使再全局let的变量，在window中也找不到！

(3). 同一作用域内，禁止同时let声明两个同名的变量。

10. 示例: 演示let的小脾气

|  |
| --- |
| // console.log(a);//undefined  // var a=10;  // console.log(a);//10  //console.log(a); //报错: Cannot access 'a' before initialization  //                      不能    访问   a   在      初始化变量 之前  //(function(){  //let a=10;    //var \_a=10;  //})();  var a=10;  (function(){    //let b=100;    var \_b=100;    console.log(window);  })();  let c=10;  let c=100; |

1. 箭头函数
2. 什么是: 专门简写function的一种新的语法
3. 何时: 今后绝大部分function，都可以用箭头函数简写。最终程序中，应该看不到任何function才对！
4. 如何:

(1). 去function，在()和{}之间加=>

(2). 如果只有一个形参，可以去掉()

(3). 如果函数体只有一句话，可以去掉{}和;

但是，如果仅剩的一句话还是return xxx，则必须去掉return

1. 示例: 简化function

使用function的代码

|  |
| --- |
| var arr=[12,23,123,32,3,1,2];  arr.sort(function(a,b){return a-b});  console.log(arr);  var str="you can you up";  str=str.replace(/\b[a-z]/ig,function(kw){    return kw.toUpperCase();  })  console.log(str);  var teachers=["亮亮","然然","东东"];  teachers.forEach(function(elem){ console.log(`${elem} - 到！`) });  var arr=[1,2,3,4,5];  var arr2=arr.map(function(elem){ return elem\*2} );  console.log(arr2);  var add=function(a,b){ return a+b };  console.log(add(3,5));//8  (function(){    var now=new Date();    console.log(`页面加载完成,at:${now.toLocaleString()}`)  })()  setInterval(function(){    console.log(`滴答`)  },1000); |

用箭头函数简化后:

|  |
| --- |
| var arr=[12,23,123,32,3,1,2];  arr.sort((a,b)=>a-b);  console.log(arr);  var str="you can you up";  str=str.replace(/\b[a-z]/ig,kw=>kw.toUpperCase())  console.log(str);  var teachers=["亮亮","然然","东东"];  teachers.forEach(t=>console.log(`${t} - 到！`));  var arr=[1,2,3,4,5];  var arr2=arr.map(elem=>elem\*2);  console.log(arr2);  var add=(a,b)=>a+b;  console.log(add(3,5));//8  (()=>{    var now=new Date();    console.log(`页面加载完成,at:${now.toLocaleString()}`)  })();  setInterval(()=>console.log(`滴答`),1000); |

1. 箭头函数的特点: 内部this指向外部的this，也就是箭头函数内外的this一致的
2. 总结：

(1). 如果当前函数没用到this，或希望函数内的this与外部this保持一致，可用箭头函数简写

(2). 如果不希望函数内的this与外部this一致时，就不能用箭头函数简写！

7. 示例: 演示箭头函数内外this一致的特点:

|  |
| --- |
| //this->window  var lilei={//不是作用域    sname: "Li Lei",    friends:["亮亮","然然","东东"],    //say:function(){    say(){//今后ES6中所有对象的方法不用加":function"    //但是这个简写不改变this！绝不等于箭头函数！      //Li Lei认识亮亮      //Li Lei认识然然      //Li Lei认识东东      //遍历lilei好友列表中每个朋友      this.friends.forEach(        f=>console.log(`${this.sname}认识${f}`)          //         希望this和say中的this保持一致都指lilei      )    }  }  lilei.say(); |

1. for of
2. 什么是: 专门简化普通for循环的新写法
3. 何时: 大部分只遍历数组的内容，不关心位置，也不改变遍历顺序的普通for循环，才能用for of简写！
4. 如何:

for(var 变量 of 数组/类数组对象/字符串){ //都是数字下标

//of会依次取出后边数组/类数组对象/字符串中每个位置上的的元素值，临时保存到of前的变量中

//of前的变量会依次得到元素值

}

1. 哪些for循环不能用for of简写

(1). 每遍历到一个元素，既关心元素值，又关心元素位置时，不能用for of。因为for of无法获得当前元素位置

(2). 从后向前反向遍历的for循环，或隔着绷着遍历的for循环，不能用for of简写

5. 示例: 使用for of遍历数组、类数组对象和字符串

|  |
| --- |
| var names=["亮亮","然然","东东"];  // for(var i=0;i<names.length;i++){  //   console.log(`${names[i]} - 到！`)  // }  for(var n of names){    console.log(`${n} - 到！`)  }  function add(){    var sum=0;    // for(var i=0;i<arguments.length;i++){    //   sum+=arguments[i]    // }    for(var n of arguments){      sum+=n;    }    return sum;  }  console.log(add(1,2,3), add(1,2,3,4,5));  var str="hello";  console.log(`hello,由以下字母组成: `)  // for(var i=0;i<str.length;i++){  //   console.log(str[i])  // }  for(var c of str){    console.log(c)  } |

6. 总结遍历:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | for | forEach | for of | for in |
| 数字下标的 | 数组 | √ | √ | √ | × |
| 类数组对象 | √ | × | √ | × |
| 字符串 | √ | × | √ | × |
| 自定义下标名称的 | 对象 | × | × | × | √ |
| 关联数组 | × | × | × | √ |

\* for in: in其实不仅遍历当前对象自己的成员，且会延着原型链向父对象继续查找所有可用的成员名

所以，一般不用for in遍历索引数组

1. 参数增强:
2. 默认值(default)

(1). 什么是: 即使调用函数时，没有为某个形参变量指定实参值，这个形参变量也有备份的默认值可用！

(2). 何时: 今后只要希望一个形参变量在调用函数时，即使没有传入实参值，也有备用的默认值使用时，就用个默认值语法。

(3). 如何: 定义函数时

function fun(形参1=默认值){

}

(4). 结果:

a. 如果将来调用函数时，给形参1提供了实参值，则使用提供的实参值

b. 如果将来调用函数时，没有给形参1提供实参值，则使用定义函数时预设的默认值作为形参变量的值。

(5). 旧js中，兼容写法:

function fun(形参1){

形参1===undefined&&(形参1=默认值)

意为: 如果形参1不是undefined时，则使用形参1的值。

如果形参1的值是undefined，则使用&&后的默认值作为代替

//复习第一阶段讲的短路逻辑和隐式转换！

}

(5). 强调: 默认值只能解决最后一个形参不确定有没有的情况！

如果多个形参都不确定有没有，应该用后边学的参数解构来解决！

(6). 示例: 使用参数默认值解决问题！

|  |
| --- |
| //function intr(msg="主人很懒，什么也没留下"){  function intr(msg){    msg===undefined&&(msg="主人很懒，什么也没留下")    console.log(`我的自我介绍是: ${msg}`)  }  intr("you can you up, no can no bibi");  intr();  //参数默认值无法解决，多个形参变量，都不确定的情况！  //想定义一个肯德基点餐的函数  function order(zhushi="香辣鸡腿堡",xiaochi="薯条",yinliao="可乐"){    console.log(`      您点的套餐是:        主食:${zhushi},        小吃:${xiaochi},        饮料:${yinliao}    `)  }  //一个都不换  order();  //三个都换  order("奥尔良烤腿堡","鸡块","奶茶")  //只想换汉堡  order("老北京鸡肉卷")  //只想换薯条！  //只想换可乐! |

1. 剩余参数(rest):

(1). 什么是:专门代替arguments的一种新的获得不确定个数实参值的新方法

(2). 为什么: arguments的缺点:

a. 单词太长！且没有意义！还不能自定义

b. 不是数组类型！无法使用数组家好的函数！

c. arguments只能所有实参值照单全收，不能有选择的收取一部分实参值！

d. ES6中的箭头函数，禁止使用arguments了！

(3). 何时: 只要想使用arguments时，都换成剩余参数语法！

(4). 如何: 定义函数时:

a. 从头到尾获得所有实参值:

function fun(...数组名){

… …

}

其中: ...的意思是”收集”，把传给函数的所有实参值都收集到...后的数组中保存！

b. 其实: 剩余参数可以只收集部分实参值

function fun(形参1, 形参2, ...数组名){

}

结果: ...数组名收集的是除了前两个形参变量对应的实参值之外的剩余实参值

(5). 好处:

a. 数组名可以自定义！

b. 是纯正的数组！可以使用数组家所有好的函数！

c. 可以有选择的获得部分实参值，而不必非要照单全收

d. ES6中的箭头函数支持剩余参数语法！！

(6). 示例: 使用剩余参数语法求和与计算薪资

|  |
| --- |
| // function add(...arr){  //   console.log(arr);  //   var result=arr.reduce(function(prev,elem){  //     return prev+elem;  //   },0);  //   return result;  // }  var add=(...arr)=>arr.reduce((prev,elem)=>prev+elem,0);  console.log(add(1,2,3), add(1,2,3,4,5));  //定义一个函数计算员工的薪资  function calc(ename,...arr){    var result=arr.reduce((prev,elem)=>prev+elem,0)    console.log(`${ename}的总工资是:${result}`);  }  calc("Li Lei",10000,1000,2000);  calc("Han Meimei", 4000,2000,3000,4000,5000) |

1. 打散数组(spread)

(1). 什么是: 将一个数组打散为多个元素值，依次传给函数的多个形参变量

(2). 何时: 只要需要多个值，但是多个值是放在数组中给的，都要先打散数组，再使用多个值

(3). 如何: 调用函数时

函数名(...数组名)

其中: ...是”打散”的意思。将数组打散为多个元素

(4). 结果: 先将数组打散为多个值，再传给函数作为实参值

(5). 其它固定套路:

a. 复制一个数组:

1). var arr2=arr.slice();

2). var arr2=[...arr] //先打散数组，将原数组中每个值放入新创建的数组中

b. 合并两个数组:

1). var arr3=[].concat(arr1,arr2)

2). var arr3=[...arr1, ...arr2] //先将两个数组打散后，将所有元素都放入新数组中！

(6). 其实...还可打散对象:

a. 浅克隆一个对象: var obj2={...obj1}; //先打散obj1对象，然后将打散后的所有属性，放入新对象中

b. 合并两个对象:

1). var obj3=Object.assign({},obj1,obj2);

先将obj1和obj2打散后，所有的属性，都放入第一个参数空对象中返回

2). var obj3={...obj1, ...obj2}; //先将obj1和obj2打散后，所有的属性，都放入新创建的空对象中返回

(7). 示例: 使用...打散数组，复制数组和对象，拼接数组和对象

|  |
| --- |
| var arr=[2,7,5,3];  //打散数组  console.log(Math.max(...arr))//7  console.log(Math.min(...arr))//2  //复制一个数组  //var arr2=arr.slice();//从头到尾选择原数组中所有元素，组成新数组  var arr2=[...arr];  console.log(arr2);  console.log(arr==arr2);//false  //拼接两个数组为一个新数组  var arr1=[1,2,3];  var arr2=[4,5,6];  //var arr3=[].concat(arr1,arr2);  var arr3=[...arr1,...arr2];  console.log(arr3);  //浅克隆一个对象:  var obj1={x:1, y:2};  var obj2={...obj1};  console.log(obj2);  console.log(obj1==obj2);//false  //合并两个对象:  var obj1={x:1,y:2};  var obj2={n:3,m:4};  //var obj3=Object.assign({},obj1,obj2);  var obj3={...obj1, ...obj2};  console.log(obj3); |

七: 解构:

1. 什么是: 仅提取出一个大的数组或对象中个别成员，单独使用！
2. 为什么:

(1). 频繁使用数组中某个元素时，总要”数组名[下标]”

(2). 频繁使用对象中的成员时，总要”对象.方法名/属性名”

3. 何时: 今后只要仅单独使用一个大的数组或对象中的一小部分内容时，都用解构！

4. 如何: 3种情况: 数组解构，对象解构，参数解构

5. 数组解构:

(1). 什么是数组解构: 仅提取出一个大的数组中个别元素，单独使用！

(2). 如何: 2步:

a. 先将要保存元素值的变量装扮成数组的样子

b. 再用等号从数组中提取对应下标位置的元素值到变量中

var [变量1, 变量2, …] = 数组

0 1

(3). 结果: 相同下标位置的变量，会接住数组中相同位置的元素值

变量1=数组[0]

变量2=数组[1]

(4). 数组解构其实就是下标对下标的赋值

(5). 固定套路: 不声明第三个变量，交换两个变量的值！

[a,b]=[b,a]

新建一个数组，把b和a的值放入新数组中

然后把a和b两个变量交换位置，装扮为一个数组

结果: a拿到的就是右边数组0位置b的值

b拿到的就是右边数组 1位置a 的值

(6). 示例: 数组解构，交换两个变量的值

|  |
| --- |
| var date=[2020,3,18];  //console.log(date[0],date[1],date[2])  // var y=date[0];  // var m=date[1];  // var d=date[2];  //想要年，月，日  // var [y,m,d]=date;  // console.log(y,m,d);  //只想要月和日  var [  , m , d]=date;  //    0  1   2  console.log(m,d)  //不声明第三个变量，交换两个变量的值  var a=5, b=8;  [a,b]=[b,a];  console.log(a,b);//8  5 |

6. 对象解构

(1). 什么是对象解构: 仅提取出一个大的对象中个别成员，单独使用！

(2). 如何: 2步:

a. 先将要保存对象中成员的变量装扮成对象的样子(同时写属性名和变量才能配对！)

b. 再用等号从对象中提取出相同属性名的成员值保存到变量中

var {属性名1: 变量1, 属性名2: 变量2, …}=对象

(3). 结果: =左边相同属性名的变量，会获得对象中相同属性名的属性值或方法.

变量1=对象.属性名1

变量2=对象.属性名2

(4). 总结: 对象解构其实就是属性名对属性名的赋值

(5). 简写:

a. 问题: 因为原对象中的属性和方法起名时已经见名知意且很简洁了。所以，我们很少需要对解构出来的属性重命名。基本上都是沿用原来的属性名。这样就导致{}中:左边和右边的名称一样，要写两遍。

b. 解决: 其实只写一个名字即可:

var {属性名1, 属性名2,…}=对象

一个名字两用！既当做配对的属性名，又当做将来单独使用的变量名。

(6). 示例: 对象解构:

|  |
| --- |
| var user={    uid:1001,    uname:"dingding",    uage:25,    login:function(){      console.log(`登录...`)    },    logout:function(){      console.log(`注销...`)    },    signup:function(){      console.log(`注册...`)    },    changePwd:function(){      console.log(`修改密码...`)    }  }  //将来都是从一个大的对象中先解构出想用的小部分，再单独使用  //只想用uname和logout  //var uname=user.uname;  //var logout=user.logout;  //:前的属性名用来配对  //:后才是要接住属性值的变量  //var {uname:uname, logout:logout}=user;  //简写:当:前后的属性名和变量名相同时，其实可以只写一个  var {uname,logout}=user;  //但是，虽然名字只写了一遍，但是一个名字两用: 即用作配对，又当将来的变量名!  console.log(`用户名:${uname}`)  logout(); |

总结: ES5中的武器:

1. 如果没有构造函数，只有一个父对象，也想创建子对象时: Object.create()
2. 替换函数中不想要的this:

(1). 在调用函数时，临时替换一次this: call或apply

a. 默认情况多数用call()

b. 多个实参值是放在一个数组中给的，需要打散数组，再逐个传参: apply()

(2). 如果希望反复调用函数，并且永久绑定this和部分实参值: bind()

3. 数组新函数:

(1). 判断:

a. 判断数组中是否所有元素都符合要求: every

b. 判断数组中是否包含符合要求的元素: some

(2). 遍历:

a. 单纯遍历，不返回新数组: forEach()

b. 保护原数组不变，返回新数组: map()

(3). 过滤和汇总:

a. 过滤: 复制出数组中符合条件的元素组成新数组: filter

b. 汇总: 对数组中的元素进行汇总: reduce

总结: this 5种:

1. obj.fun() this->obj
2. new Fun() this->new正在创建的新对象
3. fun() 或 (function(){ … })() 或 回调函数中的 this->window

因为回调函数在主函数内，相当于匿名函数自调方式执行！

1. 构造函数.prototype.共有方法=function(){ … } this->将来调用这个函数的某个子对象
2. 访问器属性中的this: 指当前访问器属性所在的对象本身(eric对象)

Object.defineProperty(eric, “eage”, {

get:function(){ return this.\_eage},

set:function(value){ this.\_eage=value }

})

总结: ES6:

1. 拼接字符串: 模板字符串
2. 声明变量: let
3. 简化function: 箭头函数
4. 简化普通for循环: for of
5. 希望不传参，形参也有默认值: function fun(形参=默认值){ … }
6. 代替arguments: function fun(...数组名);
7. 打散数组和对象: fun(...数组) [...数组,…] {...对象,…}
8. 提取出一个大的对象或数组中的个别成员，单独使用: 解构:

var [变量1, 变量2, …]=数组 或 var {属性名1, 属性名2, …}=对象

作业:

1. (必须)看小程序视频学习reduce函数高级用法:

小程序->在线->JSCORE->day06: 使用reduce统计一个字符串中每种字符出现的次数

1. (必须)看思维导图学习高频笔试题: 不声明第三个变量，交换两个变量的值，共有几种方法

思维导图->ES6->解构->如何->3种->1. 数组解构->固定套路

1. (必须)看小程序视频学习ES6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

1. (必须)看小程序视频学习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

1. (必须)看小程序视频学习高频笔试题:

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

1. (扩展)看小程序视频学习高频笔试题:

小程序->在线->JSCORE->day07:

作业: 微软数组笔试题,

闭包+面向对象,

高频笔试题 多个按钮… ,

数组去重复 Set类型