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->day07、day08

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->day07、day08

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->day07

作业: 微软 数组 笔试题 array i j

作业: 闭包+面向对象

作业: 高频笔试题 定时器setTimeout

作业: 高频笔试题 多个按钮点那个按钮显示是第几个

作业: 高频笔试题 数组去重复 set类型

是否高频: 看我的朋友圈！历届拿到高薪的师兄师姐，反馈回来的鄙视面试重点内容！

有些有心的同学，已经把朋友圈里反复出现的高频词总结出来！

3. 单词列表: 小程序->单词->JSCORE->day07、day08

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13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: 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

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. 什么是模板字符串: 支持换行，支持**动态**生成内容的新字符串

2. 为什么: 旧js中，想拼接字符串，都用+，但是+极容易和算数运算的+产生歧义

3. 何时: 今后几乎所有的字符串拼接都用模板字符串，不再用+

4. 如何:

(1). 整个字符串用一对儿反引号**`** ... **`**包裹

(2). 模板字符串内需用动态生成的内容，必须放在${}中

5. ${}中: 将来VUE框架中绑定语法和这里的规定完全一样！

(1). 能放什么？**变量，运算，三目，调用方法，创建对象，访问数组中元素**

总结: 只要有返回值的合法的变量或js表达式，都可放在${}中

(2). 不能放什么？不能放~~程序结构~~(分支和循环)，也不能放~~没有返回值的~~js表达式

6. 示例: 使用模板字符串拼接各种字符串

1\_templateString.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 price=12.5;      var count=5;      console.log(`        单价: ¥${price.toFixed(2)}        数量: ${count}      ===========================        小计: ¥${(price\*count).toFixed(2)}      `);      var sex=1;      console.log(`性别:${sex==1?"男":"女"}`);      //复习\*\*\*第一阶段\*\*\*日期:      //获得当前时间对应的ms数      //new Date().getTime()      var orderTime=1594085622575;      //将ms数转为人能看懂的日期      console.log(`下单时间:${new Date(orderTime).toLocaleString()}`);      var week=["日","一","二","三","四","五","六"];              // 0    1    2    3    4    5   6      //获得当前日期星期几对应的数字(从0开始的！星期日是一周的第一天0)      var i=new Date().getDay();//2      console.log(`今天星期${week[i]}`);      </script>  </body>  </html>  运行结果:  单价: ¥12.50  数量: 5  ===========================  小计: ¥62.50    性别:男  下单时间:2020/7/7 上午9:33:42  今天星期二 |

二. let

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

1. 什么是: 专门代替var用于声明变量的新关键词

2. 为什么: var的问题:

(1). 会被声明提前——打乱程序正常的执行顺序

(2). 没有块级作用域——代码块内的变量，很有可能影响块外的代码！

块级作用域: if,else,for,while,do while,switch等这些程序结构的**{}**，在js中都不是作用域！这些{}防不住内部的var变量被声明提前到{}外部。即使在{}内声明的变量，出了{}依然可以使用！

3. 示例: 演示var 的两个缺点:

2\_var.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 t=0;//定义全局变量t，用于累加每个函数的执行时间      function fun1(){        console.log(`fun1执行耗时0.3s`);        t+=0.3;      }      function fun2(){      //var t;// 因为fun2中已经有了局部变量t        console.log(`fun2执行耗时0.8s`);        t+=0.8; //所以0.8不会加到全局变量！而是加给局部变量t。所以，最后全局变量t少了0.8s        //将来可能会追加一段新的代码        if(false){//不会执行！但是if的{不是作用域,防不住内部的var变量被声明提前！          var t=new Date();          console.log(`新代码在${t.toLocaleString()}被执行`)        }      }      fun1();      fun2();      console.log(`共耗时${t}s`);//1.1s    </script>  </body>  </html>  运行结果:  fun1执行耗时0.3s  fun2执行耗时0.8s  共耗时0.3s |

4. 何时: **今后所有的变量都要用let声明**！

5. let的好处：

(1). 不会被声明提前——保证程序顺序执行

(2). 让代码块也变成了作用域！——保证块内的变量，无法超出块的范围影响外部！

6. 原理: let底层会悄悄的做两件事，来避免let的变量和外部变量同名！

(1). let会被自动翻译为**匿名函数自调**，有自己的作用域

(2). let的变量会被**悄悄改名**，避免重名——双保险！

7. 示例: 使用let避免不同范围之间的变量互相干扰:

3\_let.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 t=0;//定义全局变量t，用于累加每个函数的执行时间      function fun1(){        console.log(`fun1执行耗时0.3s`);        t+=0.3;      }      function fun2(){      //var t;// 因为fun2中已经有了局部变量t        console.log(`fun2执行耗时0.8s`);        t+=0.8; //所以0.8不会加到全局变量！而是加给局部变量t。所以，最后全局变量t少了0.8s        //将来可能会追加一段新的代码        if(false){//不会执行！但是if的{不是作用域,防不住内部的var变量被声明提前！          var t=new Date();          console.log(`新代码在${t.toLocaleString()}被执行`)        }      }      fun1();      fun2();      console.log(`共耗时${t}s`);//1.1s    </script>  </body>  </html>  运行结果:  fun1执行耗时0.3s  fun2执行耗时0.8s  undefined  新代码在2020/7/7 上午11:30:09被执行  100  共耗时1.1s |

8. **let小脾气:**

(1). 在全局let的变量，竟然在window里找不到！

(2). 在相同作用域内，不允许用let同时声明两个同名的变量

(3). 禁止提前使用let声明的变量

9. 示例: 演示 let的小脾气:

4\_let.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 a=10;      let b=100;      // (function(){      //   var b=100;      // })();      console.log(window);      let c=10;      //let c=100;//报错:      //Identifier 'c' has already been declared      //变量名/标识符        已经    被    声明      //console.log(d); //报错      //Cannot access 'd' before initialization      //不能    访问   d  在xxx前 初始化(首次声明并赋值)      let d=100;    </script>  </body>  </html>  运行结果: |

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: 专门简写function的定义函数的新写法

2. 为什么: 就是因为到处写function！太烦人！

3. 何时: 今后几乎所有function，都可被箭头函数代替——今后在项目中应该几乎看不到function才对！

4. 如何: 3句话:

(1). 去掉**function**，在()和{}之间加**=>**

(2). 如果形参列表**只有一个形参**，则可以**省略()**

强调: 如果**没有形参，则必须加()**

(3). 如果函数体只有**一句话**！可以省略**{}**

强调:

a. 唯一的一句话结尾，一定**不要加分号**！

b. 如果仅剩的一句话，还是return，则**必须去掉**return！

5. 示例: 使用箭头函数简写普通函数

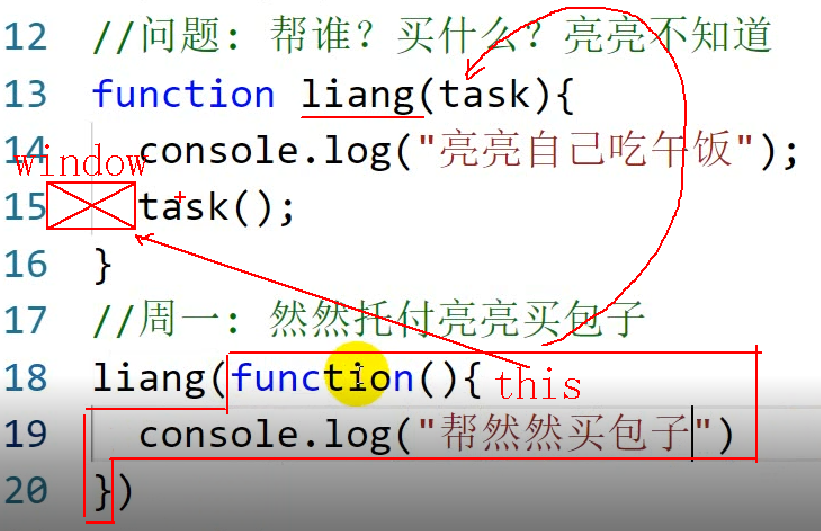
5\_arrow.html

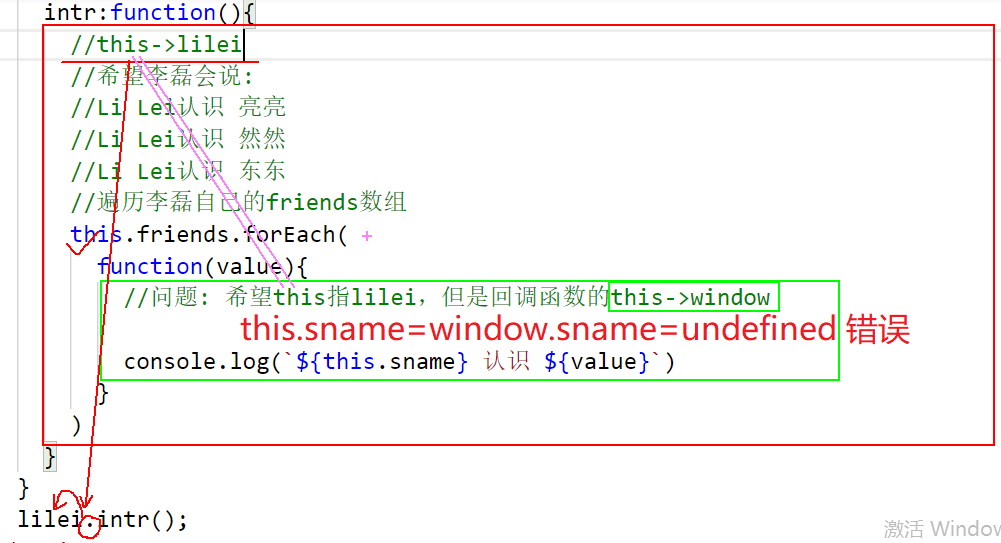
|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      // function add(a,b){      //   return a+b;      // }      // console.log(add(3,5));//8      var add=(a,b)=>a+b;      console.log(add(3,5));//8      // var arr=[23,12,2,1,123,3];      // arr.sort(function(a,b){return a-b});      // console.log(arr);      var arr=[23,12,2,1,123,3];      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();      // });      // console.log(str);      var str="you can you up";      str=str.replace(        /\b[a-z]/g,keyword=>keyword.toUpperCase());      console.log(str);      // var arr=["亮亮","然然","东东"];      // arr.forEach(function(value){      //   console.log(`${value} - 到`)      // });      var arr=["亮亮","然然","东东"];      arr.forEach(value=>console.log(`${value}-到`));      // var arr=[1,2,3,4,5];      // var arr2=arr.map(function(value){      //   return value\*2;      // });      // console.log(arr2);      var arr=[1,2,3,4,5];      var arr2=arr.map(value=>value\*2);      console.log(arr2);      // var arr=[1,2,3,4,5];      // var result=arr.reduce(function(box,value){      //   return box+value      // },0);      // console.log(result);//15      var arr=[1,2,3,4,5];      var result=arr.reduce((box,value)=>box+value,0);      console.log(result);//15      // (function(){      //   var t=new Date();      //   console.log(`网页加载完成,at:${t.toLocaleString()}`)      // })();      (()=>{        var t=new Date();        console.log(`网页加载完成,at:${t.toLocaleString()}`)      })();      // var s=5;      // var timer=setInterval(function(){      //   console.log(s);      //   s--;      //   if(s==0){      //     console.log(`boom!!!`);      //     clearInterval(timer);      //   }      // },1000)      var s=5;      var timer=setInterval(()=>{        console.log(s);        s--;        if(s==0){          console.log(`boom!!!`);          clearInterval(timer);        }      },1000)    </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  网页加载完成,at:2020/7/7 下午12:05:01  5  4  3  2  1  boom!!! |

6. 箭头函数特点:

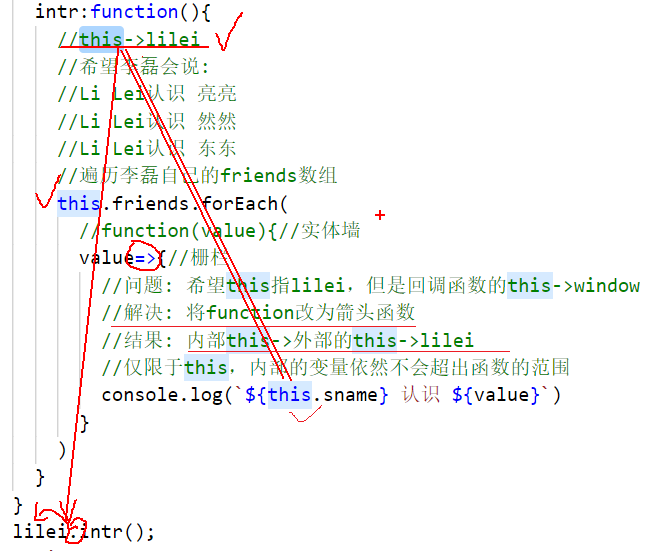
(1). 问题: **回调函数中的this，通常都指window**

因为: 回调函数在主函数中被调用时，前边往往什么都没有！





(2). 解决: **将普通的function改为箭头**



(3). 原理: **箭头函数内的this，自动与箭头函数外的this保持一致！**

7. 示例: 使用箭头函数分别简写对象方法和回调函数

6\_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>      //this->window      var lilei={//本来就不是作用域，不是实体墙！        sname:"Li Lei",        friends:["亮亮","然然","东东"],        //intr:()=>{//错误：intr内this->对象外this->window        //intr:function(){        intr(){//ES6对对象方法的简写: 可以不写":function"               //强调: ES6简写绝对不等于箭头函数，因为不影响this！this还和使用function时保持一致！          //intr内this->将来lilei.intr()中.前的lilei          //希望李磊会说:          //Li Lei认识 亮亮          //Li Lei认识 然然          //Li Lei认识 东东          //遍历李磊自己的friends数组          //如果前边intr改为=>，则这里报错:          //不能访问undefined的forEach          //this.friends=undefined          //因为this.friends=window.friends=undefined          this.friends.forEach(            //function(value){//实体墙            //问题: 希望this指lilei，但是回调函数的this->window            //解决: 将function改为箭头函数            value=>{//栅栏              //结果: 回调函数内部this->外部intr内的this->lilei              //仅限于this，内部的其它局部变量依然不会超出函数的范围              console.log(`${this.sname} 认识 ${value}`)            }          )        }      }      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 |

8. 总结：

(1). 今后，如果函数中**没用到this**，或刚好希望**内外this保持一致**时，才能**用箭头**函数简写！

(2). 今后，如果**不希望**内外this相同时！是**不能**用箭头函数简写！

比如: **对象的方法不能用箭头函数简写**（应该用ES6方法简写去掉":function"）

DOM中的事件处理函数，也不希望函数内的this与外部一致（将来讲...）



四. for of

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

1. 什么是: 单纯简化普通for循环，遍历索引数组和类数组对象

2. 为什么:

(1). 传统的for循环已经没有简化的空间

(2). forEach又只能让数组使用，类数组对象用不了！

3. 何时: 今后只要遍历索引数组和类数组对象时都可用for of

4. 如何: 元素值

↓

**for(var 变量 of 数组/类数组对象){**

**... ...**

**}**

其中: of会依次取出每个元素的**值**，保存到of前的变量中

5. 总结:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | for(var i=0; i<arr.length; i++) | forEach  (内部其实是for) | 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 | × | × | × | √ |

6. **for of不能做:**

(1). 不能获得下标位置

(2). 无法倒序遍历，或调整遍历的步长，只能从头到尾挨个遍历

(3). 也不能单纯数数，而只能遍历一个有length属性的数组或类数组对象。

7. 结论: 因为绝大多数循环，都不需要关心下标位置，不需要倒序，不需要调整步调，也不需要数数，所以for of反而比for循环和forEach用的都多！

8. 示例: 使用for of简化for循环，遍历各种东西

7\_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"];      //3种:      // for(var i=0;i<arr.length;i++){      //   console.log(`${arr[i]}-到！`)      // }      // arr.forEach(function(value){      //   console.log(`${value}-到！`)      // })      // arr.forEach(v=>console.log(`${v}-到！`));      for(var v of arr){console.log(`${v}-到！`)}      function add(){        var sum=0;        // for(var i=0;i<arguments.length;i++){        //   sum+=arguments[i];        // }        //类数组对象        //arguments.forEach(v=>sum+=v);//报错        //arguments.forEach is not a function        for(var v of arguments){sum+=v}        return sum;      }      console.log(add(1,2,3));//6      console.log(add(1,2,3,4,5));//15      var str="hello";      //       01234.length=5      for(var i=0;i<str.length;i++){//顺序遍历        console.log(str[i]);      }      for(var i=str.length-1;i>=0;i--){//倒叙遍历        console.log(str[i]);      }      for(var i=0;i<str.length;i+=2){//调整遍历的步长        console.log(str[i]);      }      // for(var v of str){      //   console.log(v);      // }      for(var i=0;i<5;i++){        console.log(i);      }      // for(var i of 5){ //报错: 5 is not iterable不可遍历      //   console.log(i)      // }    </script>  </body>  </html>  运行结果:  亮亮-到！  然然-到！  东东-到！  6  15  h  e  2 l  o  o  2 l  e  h  h  l  o  0  1  2  3  4 |

五. **参数增强**: 3个新技能

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:

(1). 什么是: 定义函数时为参数提供一个默认值，如果将来调用函数时，没有提供实参值，形参变量也有默认值使用

(2). 何时: 只要希望将来即使不传入实参值，形参变量也有默认值可用时

(3). 如何: 定义函数时:

function 函数名(形参1, ... , 最后一个形参=默认值){

//如果将来调用函数时，没有传入最后一个实参值，则最后一个形参变量也有默认值可用！

}

(4). 结果:

a. 调用时，传入了实参值，则形参变量采用传入的实参值

b. 调用时，没有传入实参值，则形参变量自动采用提前准备好的默认值

(5). 局限: 通常参数默认值只能解决最后一个形参变量不确定有没有值的情况！

如果中间某个形参不确定有没有，又想有默认值，不能用参数默认值实现的！

因为将来调用函数传参时，是不可能隔着蹦着传参的！

(6). 解决: 任意一个参数不确定，都能有默认值使用呢？！（稍后讲...）

(7). 示例: 使用参数默认值保证即使没有传入实参值，形参变量也有默认值可用

8\_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>      //定义一个函数，显示我的自我介绍！      function intr(msg="主人很懒，什么也没有留下"){        console.log(`我的自我介绍是:${msg}`)      }      //正常情况，如果传入了msg      intr("you can you up!");      //调用intr时，没有传入任何实参值      intr();      //定义一个点套餐的函数      function order(zhushi="香辣鸡腿堡",xiaochi="薯条",yinliao="可乐"){        console.log(`您点的餐是:          主食:${zhushi},          小吃:${xiaochi},          饮料:${yinliao}        `)      }      //第一个人不着急，套餐中每个东西都自己定义      order("香辣鸡腿堡","薯条","可乐");      //第二个人着急赶火车，没空挨个选择，想要套餐原装的内容      order();      //第三个人只想把最后一个可乐换成咖啡,前两项不变！      //order(,,"咖啡");//语法错误！      //第四个人只想换第二个薯条为菠萝派，主食和饮料保持不变      //order(,"菠萝派")//语法错误！    </script>  </body>  </html>  运行结果:  我的自我介绍是:you can you up!  我的自我介绍是:主人很懒，什么也没有留下  您点的餐是:  主食:香辣鸡腿堡,  小吃:薯条,  饮料:可乐    您点的餐是:  主食:香辣鸡腿堡,  小吃:薯条,  饮料:可乐 |

2. 剩余参数:

(1). 问题: ES6中箭头函数中**禁止**使用arguments！

(2). 解决: 用ES6中的剩余参数语法代替arguments

(3). 什么是剩余参数: 专门代替arguments来获得不确定个数的实参值的新语法

(4). 何时: 今后只要在es6代码中，遇到不确定参数个数的情况，都用剩余参数语法代替arguments

(5). 如何: 定义函数时:

function 函数名(形参1, 形参2, **...数组名**){

//...数组名，在将来调用函数时，会自动收集除前两个形参之外的，多余的所有剩余参数值，保存进一个自定义名称的数组中！

}

(6). 剩余参数的数组和arguments的差异:

a. 类型:

1). 剩余参数的数组是纯正的**数组**类型，可以使用数组家所有的函数

2). arguments是**类数组**对象，不是数组家孩子，所以不能使用数组家函数

b. 获取哪些实参值:

1). 剩余参数的数组之前可以定义其它形参，而剩余参数的数组仅获取其它形参不要的多余的剩余参数值

2). arguments只能获得所有实参值！不能有所选择！

(7). 示例: 用剩余参数代替arguments实现求和、计算总工资的功能

9\_rest.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(){      //用箭头函数简写add函数      //       可以改名      var add=(...arr)=>{      //...收集, arr是数组名      //(将剩余实参值都)收集到arr(数组中保存)！        //用for of:        //var sum=0;        //for(var v of arguments){//报错: arguments is not defined——箭头函数中不允许使用arguments了！        //for(var v of arr){        //  sum+=v;        //}          //用reduce        var sum=arr.reduce((box,v)=>box+v,0);        return sum;      }      //var add=(...arr)=>arr.reduce((box,v)=>box+v,0);//语法糖      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      //用户至少输入自己的员工姓名，但是每个员工的工资项目数不一样      //              必须  其它剩余      function jisuan(ename,...arr){        console.log(arguments);//含员工姓名        console.log(arr);//不含员工姓名        console.log(`${ename}的总工资是:${          arr.reduce((box,v)=>box+v,0)        }`);      }      jisuan("Li Lei",10000,1000,2000);      //      ename  ...arr[10000,1000,2000]      jisuan("Han Meimei",1000,2000,3000,4000,5000)      //      ename     ...arr[1000,2000,3000,4000,5000]    </script>  </body>  </html>  运行结果:  6  15  Arguments(4) ["Li Lei", 10000, 1000, 2000, ... ]  (3) [10000, 1000, 2000]  Li Lei的总工资是:13000  Arguments(6) ["Han Meimei", 1000, 2000, 3000, 4000, 5000, ...]  (5) [1000, 2000, 3000, 4000, 5000]  Han Meimei的总工资是:15000 |

3. 打散数组:

(1). 问题: apply()，虽然可以打散数组再传参，但是apply()的主要作用是替换this，捎带着打散数组。而实际开发中，有很多情况只单纯打散数组再传参，不需要替换this！用apply()，就很不方便！

(2). 解决: 今后只要单纯打散数组，再传参时，与this无关时，都首选ES6的打散数组语法

(3). 如何: 调用函数时:

**函数(...数组名)**

打散！

(4). 总结: **定义**函数时...是**收集**，**调用**函数时...是**打散**

(5). 示例: 使用Math.max和Math.min获取一个数组中的最大值和最小值

10\_spread.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(2,7,1,5));//7      console.log(Math.min(2,7,1,5));//1      var arr=[2,7,1,5];      //想获取数组中的最大值和最小值，又不愿意自己写遍历！      //console.log(Math.max(arr)); //NaN max不支持数组！      //尝试用apply先打散数组，再传给Math.max()      console.log(        //             替换this的对象, 要打散的数组        Math.max.apply(    null     ,   arr) //错！        //因为本例和this无关        //所以第一个实参值，填什么都行！！！——别扭！      );      //ES6的spread语法: 先打散arr数组，再传参  **console.log(Math.max(...arr));**//7      console.log(Math.min(...arr));//1    </script>  </body>  </html>  运行结果:  7  1  7  7  1 |

(6). 其实: ...有很多极其简洁的语法糖！

11\_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>      //**复制一份数组:**      var arr=[1,2,3];  **var arr2=[...arr];**            //[]创建新空数组              //...arr 先打散原数组，再传值，保存进新数组      console.log(arr2);      console.log(arr==arr2);//false      //**合并两个数组或多个数组**      var arr1=[1,2,3];      var arr2=[5,6,7];  **var arr3=[...arr1,4,...arr2,8];**      console.log(arr3);      //**克隆一个对象**:      var lilei={        sname:"Li Lei",        sage:11      };  **var lilei2={...lilei};**              //{}创建空对象              //...lilei：先打散李磊对象为多个属性，再传入新对象中保存      console.log(lilei2);      console.log(lilei2==lilei);//false      //**合并两个对象或多个对象以及属性值**      var obj1={x:1,y:2};      var obj2={m:4,n:5};  **var obj3={**  **...obj1,**  **z:3,**  **...obj2,**  **o:6**  **};**      console.log(obj3);    </script>  </body>  </html>  运行结果:  (3) [1, 2, 3]  false  (8) [1, 2, 3, 4, 5, 6, 7, 8]  {sname: "Li Lei", sage: 11}  false  {x: 1, y: 2, z: 3, m: 4, n: 5, …} |

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循环遍历原数组中每个元素:

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). 今后，只要拼接字符串，都用模板字符串:

a. 整个字符串包裹在一对儿反引号`...`中

b. 字符串中动态生成的内容必须放在${}里

c. ${}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). 今后，声明变量都用let代替var

a. let的好处: 阻止声明提前，让代码块也变成块级作用域

b. let的原理: 匿名函数自调+自动修改变量名

c. let的小脾气: 禁止重复声明; 禁止提前使用; 在全局声明也不保存在window中

(3). 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  (内部其实是for) | 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 }

|  |
| --- |
| 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(){ ... })() 或 多数回调函数 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 |

今日对应小程序视频列表:

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

作业:

1. 复习今日小程序视频：小程序->首页->JSCORE->day07, day08

2. 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

4. 预习:

小程序->在线->JSCORE->day07 6. 解构(上)...

小程序->在线->JSCORE->day08 1. 解构(下)...

2. class共有属性 extends es6 两种类型间的继承

3. 回调低于callback hell和promise解决回调地狱问题