正课:

1. ES5

2. ES6

1. ES5

call/apply/bind

什么是: 替换函数中不想要的this为指定的对象

何时: 如果函数执行时，其中的this不是想要的，都要替换this为想要的对象

如何:

1. call/apply

在本次调用函数时，临时替换函数中的this为指定对象

本次调用后，替换失效。

call: 任意函数.call(任意对象, 实参值列表...)

让任何一个对象，可临时调用任何一个不相干的函数，并传入实参值列表。

比如: calc.call(lilei,10000,2000,3000);

用lilei调用calc函数，并传入10000,2000,3000三个参数值。

apply: 如果实参值列表是以数组形式给的，则必须将call

换成apply

如何: var arr=[10000,2000,3000] //实参值列表

calc.apply(lilei, arr)

用lilei调用calc，但是会打散数组参数为单个值

比如: arr=[10000,2000,3000]

calc.apply(lilei, arr )

打散数组 ↓ ↓ ↓

等效于: calc.call(lilei, 10000,2000,3000)

结果: this base bonus1 bonus2

2. bind:

什么是: 基于原函数，创建一个新函数，并永久绑定this

何时: 如果函数中的this，需要反复被替换，就用bind创建一个新的专属的函数。永久绑定this。

如何: var 新函数=原函数.bind(对象, 要绑死的实参值,...)

比如: var calc\_l=calc.bind(lilei, 10000)

创建一个和calc一模一样的新函数calc\_l，并永久替换calc\_l中的this为lilei对象，并永久替换第一个参数base为10000。

好处: 今后反复使用calc\_l不用再反复传入lilei和10000.

强调: bind仅创建新函数，不执行函数

要执行函数: calc\_l(1000,2000)

//bonus1, bonus2

强调: 用bind创建的新函数中的this，不可能再被别人用call替换。因为bind后的函数中的this已经被替换成固定的对象了。所以再用call/apply已无this可换

总结:

当函数中的this不是想要的，想换成想要的

就用call/apply/bind

1. 当仅临时替换本次调用函数中的this时——call

当实参值列表是数组形式时，必须将call换成apply，自动打散数组为单个值

2. 如果一个函数需要反复替换this为一个指定的对象时，就用bind创建一个专属的新函数，永久绑定this为指定的对象，同时可绑定部分固定的参数值。

数组API:

ES5中为数组扩展了很多新的API

1. 数组中也有indexOf方法，查找一个指定元素的位置

用法同字符串.indexOf完全一样:

var i=arr.indexOf(要找的元素, [开始位置])

// 3 , 5

//从5位置开始找下一个3

在arr数组中，从starti位置开始，查找下一个指定元素的位置i

其中: starti可省略，默认从0位置开始找

如果找到匹配的，返回元素的下标位置

如果没找到，返回-1

比如:

var str="1234321";

// 0123456

console.log(

//在arr中找第一个3的位置

str.indexOf(3/\*,0\*/),//2

//在arr中从3位置开始找下一个3的位置

str.indexOf(3,3),//4

//在arr中从5位置开始找下一个3的位置

str.indexOf(3,5),//-1 //找不到了

//在arr中找第一个5的位置

str.indexOf(5) //-1 //找不到

);

var arr=[1,2,3,4,3,2,1]

// 0 1 2 3 4 5 6

console.log(

//在arr中找第一个3的位置

arr.indexOf(3/\*,0\*/),//2

//在arr中从3位置开始找下一个3的位置

arr.indexOf(3,3),//4

//在arr中从5位置开始找下一个3的位置

arr.indexOf(3,5),//-1 //找不到了

//在arr中找第一个5的位置

arr.indexOf(5) //-1 //找不到

);

2. 判断: 判断数组中的元素是否符合要求

arr.every(条件) 判断数组中是否所有元素都符合条件

每个

arr.some(条件) 判断数组中是否包含符合条件的元素

一些

如何: 仅以every为例

var bool=arr.every(function(elem, i, arr){//规定

//elem: 自动获得当前位置上的元素值

//i: 自动获得当前位置下标i

//arr: 自动获得.前的数组对象

return 判断条件

//如果当前元素值elem，符合判断条件要求，返回true

//如果当前元素值elem，不符合判断条件，返回false

})

执行原理: every会自动遍历arr中每个元素，每遍历一个元素，就用回调函数在当前元素上执行一次。每次调用是都会自动给回调函数传入当前元素值、当前位置和当前数组对象。中途every会记录每个元素判断后得到的结果。最后，只有所有元素判断的结果都为true，整体才返回true。只要有一个元素的判断结果返回false，整体就返回false

比如:

var arr1=[1,2,3,4,5];

var arr2=[2,4,6,4,2];

//判断哪个数组全是偶数

console.log(

arr1.every(function(elem){

return elem%2==0

}),//false

arr2.every(function(elem){

return elem%2==0

})//true

);

再比如:

//判断哪个数组是升序

//判断每个元素都小于等于下一个位置的元素

console.log(

arr1.every(

//回调函数会被every自动拿到每个元素上执行一次

function(elem,i,arr){

//elem: 自动收到当前正在遍历到的元素值

//i:当前正在遍历的位置

//arr: 当前数组对象

//如果当前元素<=当前数组的下一个位置

return i<arr.length-1?elem<=arr[i+1]:true//判断条件

//如果i没有到最后一个元素，就按规矩比较。如果已经到最后一个元素了，就不比较直接返回true！

}

),//true

arr2.every(

function(elem,i,arr){

return i<arr.length-1?elem<=arr[i+1]:true

}

)//false

);

arr.some()和arr.every()用法完全一样，只不过不要求所有元素都满足条件才返回true。而是只要有一个元素满足条件，就返回true。

3. 遍历:

1. arr.forEach(): 仅遍历，对原数组中每个元素执行相同操作

何时: 如果仅遍历数组没有返回值，或仅对原数组中的元素执行修改操作，不需要返回新数组时。

如何:

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

对elem、i、arr执行操作!

})

比如:

var names=["亮亮","然然","东东","涛涛","磊磊"];

/\*for(var i=0;i<names.length;i++){

alert(names[i]+" - 到！");

}\*/

// 对(names中)每个元素，执行回调函数

/\*names.forEach(

//回调函数会在每个元素上执行一次！

function(elem,i,arr){

//elem:自动收到当前元素的值

//i:自动收到当前位置i

//arr:自动收到当前数组对象

alert(elem+" - 到！");

//强调: 不需要return!

}

)\*/

//names.forEach(elem=>alert(elem));

2. arr.map(): 遍历取出原数组中每个元素，执行相同操作后，放入一个新数组中返回。原数组保持不变

何时: 如果希望保护原数组不变，而返回新数组时

如何:

var 新数组=arr.map(

//创建一个新的空数组: newArr:[ ]

//自动遍历原数组中每个元素

//每遍历一个元素，就调用一次回调函数

function(elem,i,arr){

return 新值

}

//每次执行完回调函数后，会将返回的新值，自动放入新数组中的相同位置

//newArr[i]=返回新值

//当遍历完成后，将新数组返回！

//return newArr;

)

比如:

var nums=[1,2,3,4,5];

var evens=nums.map(

//var newArr=[];

function(elem){

return elem\*2;

}//newArr[i]=返回值

//return newArr;

); //↓

//evens=newArr;

console.log(String(nums));

console.log(String(evens));

4. 过滤和汇总:

过滤: filter

什么是: 复制出原数组中符合条件的元素，组成新的子数组返回。原数组不变。——为什么: 因为未经允许不能擅自修改原数组。

何时: 只要希望仅获得数组中符合条件的元素时

如何:

var subArr=arr.filter(

//var subArr=[];

//filter会拿着回调函数去每个元素上检查一次

function(elem,i,arr){

return 判断条件

}

//如果当前元素检查结果返回true，则复制一份，追加到subArr中。不保证是原数组中相同的位置

//if(return true) subArr.push(elem)

//遍历结束后

//return subArr

)

比如:

var arr=[1,2,3,4,5];

var evens=arr.filter(

//先创建空数组等着接符合条件的元素

//var subArr=[]

//filter会拿着回调函数去每个元素上判断一次

function(elem){

//只要当前元素是偶数

return elem%2==0//就放入新数组

}

//如果当前元素判断结果为true，就追加到新数组

//if(return true) subArr.push(elem)

//遍历结束，返回subArr

)

console.log(arr);

console.log(evens);

汇总: 对数组中的元素内容，进行统计，返回统计结果

如何: var result=arr.reduce(function(prev,elem,i,arr){

//prev: 截止到当前元素之前临时的汇总值

//elem、i、arr同上

return 将当前元素值，汇总到prev中，形成新的汇总结果

}, 起始值 )

比如:

var class1=[1,3,5,7,9];

var result=class1.reduce(

function(prev,elem){

return prev+elem;//将当前值汇总到临时汇总值中

},

0//从0开始累加

);