正课:

ES5:

1. \*\*\*call,apply,bind

2. 数组API

3. 严格模式

ES6: let, rest, default——已讲过

模板字符串, for...of, 块级作用域, 箭头函数, class

ES5:

1. \*\*\*call,apply,bind

共同: 都是为了替换函数中不想要的this为想要的对象

何时: 只要函数中的this不是想要的，就要用call/apply/bind替换

差别:

call,apply:调用函数,临时替换函数中的this为指定对象

差别: call要求传入函数的参数列表，必须单独传入

apply要求传入函数的参数列表，必须放在一个数组中整体传入。apply可打散数组类型参数

bind: 创建一个和原函数完全相同的新函数，但提前永久绑定this为指定对象。还可提前永久绑定部分参数值

何时: 如果希望替换函数中的this，但暂不调用时

今后，只要替换回调函数中的this，都用bind

2. 数组API:

已学过: .indexOf

问题: 仅简化了代码，并没有提升效率

判断:

.every: 判断数组中每个元素是否都满足要求

.some: 判断数组中是否包含满足要求的元素

如何: var bool=arr.every(function(val,i,arr){

//val当前正在遍历的元素值

//i 当前正在遍历的位置

//arr 当前正在遍历的数组对象

return 判断条件

})

如果arr中每个元素，都满足判断条件的要求，every才返回true，否则，只要有一个元素不满足要求，就返回false

遍历: 对数组中每个元素执行相同操作

.forEach: 对原数组中每个元素执行相同操作

直接修改原数组

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

arr[i]=新值;

})

.map: 取出原数组中每个元素值，执行相同操作后，放入一个新数组中返回

不直接修改原数组，仅返回新数组

arr.map(function(val,i,arr){

return 新值;

})

共同的缺点: 不支持break! 必须遍历所有元素

过滤和汇总:

.filter: 选取出原数组中符合条件的元素组成新数组

如何: var subs=arr.filter(function(val,i,arr){

return 判断条件

})

筛选出arr中符合判断条件的元素，保存到subs

强调: 原数组保持不变，仅复制出符合条件的元素的副本

.reduce: 遍历数组中每个元素，将每个元素值汇总出一个最终结论.

如何: var sum=arr.reduce(function(prev,val,i,arr){

prev //获得截止到当前元素之前的临时汇总值

return prev+val

},base) //从base开始累加

3. 严格模式:

什么是: 比普通js运行模式更严格的模式

何时: 今后所有的项目都应该在严格模式下开发

如何: 2个范围:

1. 整个<script>或js文件范围内启用严格模式:

在<script>或js文件的顶部加入"use strict";

如果从0开始开发的一个新项目，就在整个<script>或js文件启用严格模式

2. 仅在一个函数内启用严格模式

在函数定义内部的顶部加入"use strict";

如果一个旧项目，向严格模式迁移时，应该逐个函数向严格模式迁移。

要求:

1. 禁止给未声明的变量赋值

2. 静默失败升级为错误

3. 禁止使用arguments,arguments.callee...

ES6:

1. 模板字符串: 简化字符串拼接

整个模板字符串用` `包裹

模板字符串本身支持换行,省略\n

模板字符串中可执行任意表达式,但表达式必须用${...}包裹

2. for...of: 简化版的for循环:

为什么:

for循环太长

forEach 又不能中途停止,只能被数组使用

for...in key返回的是字符串

何时: 今后凡是遍历数字下标的数组，类数组对象，字符串都可用for...of

如何: for(var val of arr){

val 直接获取当前元素值

}

缺点:

1. 无法获得下标

2. 无法直接操作原数组中的元素

3. 块级作用域:

什么是: 只要一个{}都是一级块级作用域！包括:

if...else if...else , for, while/do...while, switch,...

如何: 所有变量都用let声明:

1. 阻止声明提前

2. 仅在块级作用域中有效

4. 箭头函数Arrow: 简化所有回调函数

基本: 去function，添=>

更简化: 如果函数体只有一句话，可省略{}

更简化: 如果仅有的一句话还是return，可省略return

如果只有一个参数,可省略()

如果没有参数, 必须保留空的()

强调: 箭头函数中的this是内外一致的

5. class: 简化OOP

创建一个类型:

class类型名{

constructor(属性参数列表){

this.属性名=属性参数;

... ...

}

方法名 (){

...this.属性名...

}

}

访问器属性:

class 类型名{

constructor(){

this.\_受保护的属性=值;

}

get 访问器属性(){return this. \_受保护的属性}

set 访问器属性(val){

验证val，并赋值到this. \_受保护的属性

}

}

继承: 子类型继承父类型:

class sub extends 父类型{

constructor(...){

super(...)

}

子类型方法...

}

**class** Father {  
 constructor(name, age, saler = 12000) {  
 **this**.**name** = **'yys'**;  
 **this**.**\_age** = 30;  
 **this**.**saler** = saler  
 }  
 **get** age() {  
 **return this**.**\_age** }  
 **set** age(val) {  
 **if** (val >= 18 && val <= 60) {  
 **this**.**\_age** = val  
 }  
 **else** {  
 **throw new *Error***(**'年龄在18~60之间'**)  
 }  
 }  
 count() {  
 **this**.**saler** += 100  
 }  
}  
**const *obj*** = **new** Father(**'胜'**, 19, 15000)  
***obj***.age = 18  
***console***.log(***obj***)  
  
**class** Child **extends** Father{  
 constructor(height) {  
 **super**()  
 **this**.**height** = height  
 }  
}  
**const *obj2*** = **new** Child(165)  
***obj2***.count()  
***console***.log(***obj2***)