正课:

1. \*\*\*\*ES5

\*\*\*\*保护对象: 保护属性，防篡改

Object.create:

bind:

数组API:

严格模式:

2. ES6

模板字符串:

let

箭头函数

for of

\*\*\*\*class

1. \*\*\*\*ES5:

ECMAScript: ECMA组织制定的JavaScript语言的国际标准，所有浏览器厂商遵照实现。规定了JS语言的核心语法。

保护对象:

为什么: JS中的对象可随意修改属性值，可随意添加删除属性。

如何保护:

保护属性: 保护对属性值的修改

对象属性分为:

命名属性: 可直接用.访问到的属性

数据属性: 直接存储属性值的属性

如何保护: 四大特性:

value: 实际存储属性值

writable: 控制属性是否可修改

enumerable: 控制属性是否可被遍历

仅控制遍历，无法控制用.访问

configurable: 控制是否可删除属性

控制是否可修改其他两个特性

强调: configurable经常作为前两个属性的双保险，且一旦设为false，不可逆！

如果查看四大特性:

Object.getOwnPropertyDescriptor(obj,"属性名")

如何修改四大特性:

Object.defineProperty(obj,"属性名",{

特性:值,

特性:值,

... ...

})

问题: defineProperty一次只能修改一个属性

解决: 同时修改多个属性:

Object.defineProperties(obj,{

属性名:{ 要修改的特性 },

属性名:{ 要修改的特性 },

... : ...

})

问题: 无法使用自定义逻辑保护属性

解决:

访问器属性: 不直接存储属性值

仅提供对其他数据属性的保护

何时: 只要用自定义逻辑保护属性时

如何: 2步:

1. 定义一个隐藏的数据属性实际存储属性值

2. 添加访问器属性保护隐藏的数据属性:

Object.defineProperty(obj,"属性名",{

get(){//在试图获取属性值时自动调用

//返回受保护的数据属性值

},

set(val){//在试图修改属性值时自动调用

//参数val会自动获得要修改的新值

//如果验证val符合规则

//才将val赋值给受保护的属性

//否则

//报错！

},

enumerable:true,

configurable:false

})

如何使用访问器属性: 同普通属性用法完全一致

其中赋值时，自动调用set，取值时自动调get

内部属性: 不能用.直接访问的隐藏属性

\_\_proto\_\_

防篡改: 保护对对象结构的修改

3个级别:

1. 防扩展: 禁止添加新属性

Object.preventExtensions(obj)

原理: 每个obj内部都有一个隐藏属性:

Extensible，默认为true

preventExtensions将obj的Extensible改为false

2. 密封: 在防扩展基础上，进一步禁止删除现有属性

Object.seal(obj)

原理: 修改obj的Extensible为false

将所有属性的configurable都改为false

3. 冻结: 在密封基础上禁止修改任何值

Object.freeze(obj)

原理: 修改obj的Extensible为false

将所有属性的configurable都改为false

还将所有属性的writable都改为false

Object.create(): 可直接用一个父对象创建一个子对象。

如何: var child=Object.create(father,{

自有属性:{

value:值,

writable:true,

enumerable:true,

configurable:true,

},

... : {

...

}

});

强调: 只要添加到对象中的属性，四大特性默认为false，必须显式写为true。

原理: 1. 创建一个空对象

2. 让新对象自动继承father

3. 为新对象扩展新的自有属性

何时: 如果没有构造函数，只有父对象，也想创建子对象时。

bind: 基于一个现有函数，创建一个新函数，并永久绑定this为指定对象。

何时: 只要回调函数中的this不是想要的，就可用bind替换。

如何: var newFun=fun.bind(obj)

基于现有函数fun，创建一个功能完全相同的新函数newFun，但永久绑定newFun中的this为obj

鄙视: call,apply,bind

call和apply: 强行\*调用\*一个函数并\*临时\*替换函数中的this为指定对象。

差别: 仅在参数上:

call要求传入函数的参数，必须单独传入逗号分隔

apply要求传入函数的参数，必须放入数组中整体传入。\*apply可自动打散数组类型的参数\*

bind: 基于一个现有函数\*创建\*一个功能完全相同的新函数，并\*永久\*绑定this为指定对象，还可永久绑定部分固定的参数值。

强调: 用bind绑定的this，不能再被call/apply替换

总结: 只要函数中的this不是想要的:

call/apply: 如果立刻执行函数

call: 如果参数单独传入

apply: 如果参数放在数组中传入

bind: 希望创建一个新函数作为回调函数给别人用时

数组API:

判断: 判断数组中的元素是否符合要求

返回值: bool

1. every: 判断数组中是否每个元素都满足要求

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

return 判断条件

})

2. some: 判断数组中是否包含满足要求的元素

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

return 判断条件

})

强调: 数组API的回调函数中this默认->window

所以，不能用this指代当前元素值

比如用arr[i]或val

遍历: 对数组中每元素执行相同的操作:

1. forEach: 对原数组中每个元素执行相同的操作

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

对当前元素执行的操作

})

2. map: 取出原数组中每个元素，执行相同操作后，放入新数组返回

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

return 对当前元素执行操作后的返回值

})

过滤和汇总:

1. 过滤: 复制出原数组中符合条件的元素组成新数组

var subArr=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值，放入新数组返回。

2. 汇总: 将数组中所有元素，统计出一个汇总结果

返回值: 值

var r=arr.reduce(function(prev,val,i,arr){

//prev: 截止目前的临时汇总值

return prev+val

},startVal);

将arr数组中每个值累加后，求和。

其实: reduce不一定非要从0开始累加

可从任意startVal开始累加

严格模式:

什么是: 比普通js运行模式更严格的模式

为什么: 解决普通js运行模式中固有的弊端和缺陷

何时: 2种:

1. 新项目, 建议必须使用严格模式

2. 旧项目, 建议逐个模块向严格模式迁移

如何: 2种:

1. 整个代码段启用严格模式:

在script元素或js文件的开头加入:"use strict";

2. 仅对单个函数启用严格模式

在function内，函数体的顶部加入:"use strict";

要求:

1. 不允许对未声明的变量赋值

2. 静默失败升级为错误

3. 不建议使用arguments和arguments.callee实现递归

2. ES6:

模板字符串: 对字符串拼接的简化:

何时: 只要字符串拼接时，需要动态执行表达式

如何: `反引号——ESC键的正下方

模板字符串必须用``包裹

``中支持: 换行, 变量, 表达式

强调: 每个变量和表达式都要用${...}包裹

let: 创建一个仅在当前块内有效的变量

为什么:

1. js中没有块级作用域，导致块内的变量会污染外部。

2. js的声明提前会打乱程序的正常执行顺序

何时: 今后强烈建议用let代替var

如何:

强调: let必须配套严格模式使用

1. 解决声明提前: 检查let a之前不允许提前使用a

变相强制将所有变量的声明放在当前作用域的顶部

2. 解决块级作用域:

let声明的变量，仅在当前{}内有效，出了块失效。

箭头函数: 简化回调函数:

何时: 今后，几乎所有的回调函数都可用箭头函数简化

如何:

1. 所有回调函数都可: 去function改=>

2. 如果函数体只有一句话: 可省略{}

如果这一句话还是return，可省略return

3. 如果只有一个参数: 可省略()

但是,如果没有参数，必须保留空()

更大用途: 箭头函数内外共用同一个this——取代bind

特殊: 如果不希望内外共用this，就不能用箭头函数

比如事件处理函数:

elem.addEventListener("click",function(){this->elem})

elem.addEventListener("click",()=>{ this->不是elem})

变通解决:

elem.addEventListener("click",e=>{ e.target->elem})

for of: 简化普通for循环:

何时: 直接获得每个元素值时

如何: for(var val of arr){ val//自动获得当前元素值}

其中: of会自动取出arr中每个元素的值，保存到val

局限: 1. of只能依次遍历每个元素

2. 按值传递,无法修改原数组中的值

如果要直接修改原数组中的值，还要用普通for循环

3. 只能遍历索引数组和类数组对象。不能遍历关联数组和对象。

如果遍历关联数组和对象，还要用for in循环

\*\*\*\*class: 为了简化面向对象

1. 定义类型:

一个类型的所有属性和方法都集中包含在class{}中

类型名提升到class之后，称为类名

构造函数一律用constructor定义,其余保持不变

共有方法可直接放入class内，等效于自动放入原型对象中。

class Flyer {

constructor(fname,speed){

this.fname=fname;

this.speed=speed;

}

fly (){//Flyer.prototype.fly

console.log(...);

}

}

2. 继承:

//让Plane继承Flyer

class Plane extends Flyer{

constructor(fname,speed,score){

//super()指代父类型构造，且自动替换this

super(fname,speed);

this.score=score;

}

getScore (){...}

}